EventChain: A Distributed History Without a Distributed System

Raam Roche

2026-05-01

Table of contents

1 Executive Summary 5

2 Reconciliation After the Fact 7
- 2.1 Each Party Keeps Its Own Records . . . 7
- 2.2 Why Now . . . 7
- 2.3 Who Keeps the History . . . 8
  - 2.3.1 Why Blockchains Stalled . . . 9
- 2.4 Five Requirements . . . 10

3 Consensus by Comparison 12
- 3.1 What You Get . . . 12
- 3.2 What It Costs . . . 14
- 3.3 What Remains Trusted . . . 18
- 3.4 Where It Breaks . . . 18
- 3.5 When It Fits . . . 18

4 How EventChain Works 19
- 4.1 Append-Only File . . . 19
- 4.2 Payload Hash . . . 21
- 4.3 What a Signature Proves . . . 21
- 4.4 Append Protocol . . . 23
  - 4.4.1 Hub Heartbeat . . . 24
- 4.5 Scale and Ordering . . . 24
- 4.6 Cryptographically Bound Contracts . . . 26
- 4.7 Access to Payloads . . . 26
- 4.8 OpenTimestamps . . . 26
- 4.9 Verification . . . 27
  - 4.9.1 What Each Check Needs . . . 27

5 Open Protocol 28
- 5.1 Protocol Stewardship . . . 28
- 5.2 Bootstrapping . . . 29

6 Conclusion 31

Appendix: Worked Examples 32
Every Reading Behind the Bill . . . 32
Every Estimate Behind a Claim . . . 33
Every Document Handed to an AI Supplier . . . 33
Every Check Behind a Mortgage . . . 34
Every Handoff of a Part . . . 35

Appendix: Specification 36
Entry Specification . . . 36
WebAuthn Assertion Checks . . . 38
Heartbeat Encoding and Retrieval . . . 39
Submission, Receipt and Refusal . . . 39
CBC and Correction References . . . 41
Corrections . . . 41
Key Registries . . . 42
Payload Access and Export . . . 43
Verifier Outcomes for Heartbeat Evidence . . . 43

Appendix: Comparison Tables 44
Standards Lineage . . . 44
Dishonest Hub . . . 46
Trust Boundary . . . 47
Failure Modes . . . 47
Choosing an Approach . . . 50
Deployment Dependencies . . . 51

References 53

Abstract

Today each party to a process keeps its own records of it and reconciles them with the others' after the fact. When the records disagree, the party that cannot prove what happened carries the loss. Blockchains promised one shared history, but a blockchain agrees on each entry before it writes it, so every member had to build and pay for its system before the first entry was worth writing, and several consortia closed before they reached that point.

This paper defines EventChain, a protocol and file format for that shared history. It states what each guarantee assumes, compares the guarantees and costs with those of public and permissioned blockchains on dated published figures, and takes five processes through it in worked examples. A reference implementation accompanies the specification.

EventChain allows different holders to know that every other holder checking the same chain of events holds the same history through the entry it names. That is the certainty a blockchain promises, and no party has to join a network to get it. An entry costs one signature and one write. One company starts alone, the work grows with the number of events, and adding a party adds a copy of the file.

What becomes worth recording is every reading, revision, approval and handoff that a blockchain made too expensive to record one at a time. A dispute then starts from evidence both sides already hold, and a payment or an approval can follow the evidence the moment it arrives. A valid chain shows which key signed what, and in what order. Whether a statement was true stays with its signer, and what to do about it stays with the application that acts on the evidence. Where strangers settle payments with one another, a blockchain's round of votes remains the right tool.

Revision **91a4776** — October 2026

1 Executive Summary

Suppose every reading, estimate, approval and handoff is signed and attested with a personal key, as it is made. An invoice arrives with its deliveries already agreed. A payment or an approval follows the evidence the moment it arrives, because the other side already holds it. The content behind each entry stays encrypted and readable only by those the parties allow. Software the parties allow can act on that content as soon as the evidence meets its rules, with no person checking each step again. Nothing waits for reconciliation. There is nothing left to reconstruct. This is in essence, the promise of enterprise blockchain technology.

A blockchain agrees on each entry at the moment of writing, by a round of votes among its members. Every member must therefore have built and paid for its system before the first entry is worth writing, and every entry afterwards runs through that round. This prerequisite proved to be an obstacle to B3i, a consortium of insurers which closed in 2022. The chief executive of Swiss Re, a member, put it this way: “it would need all insurance companies to basically create smart contracts at the beginning” (Howard 2022). Everyone had to join before anything worked. With EventChain, one company starts recording on its own. The others take a copy when they need it. TradeLens wanted to connect ten ocean carriers. But rivals objected to putting their data on a platform that a competitor controlled (Hill 2018; United Nations ESCAP 2025) an couldn’t get off the ground. The agreement needs a place to happen, and at TradeLens that place belonged to a rival. With EventChain, each party checks its own copy. The service that distributes the history, the Hub, never sees what any document says. In 2022 ASX wrote off A$245–255 million. An independent review found little value in running all the business logic on a blockchain, because ASX already maintained the data’s integrity (Accenture 2022; ASX Limited 2022). Every entry waited for an agreement ASX did not need. In EventChain an entry costs one signature from whoever produced the event, and one write on a simple plaintext file. The signature is the approval or the reading itself. Energinet designed its energy-certificate system to combine millions of certificates into a single blockchain entry, citing the blockchain’s limited scalability and its cost (Energinet 2023). When every entry costs everyone’s agreement, readings come in bundles. This substantially reduced costs but introduced a trade-off: certificates could only be independently verified against the blockchain after their batch had been committed. In EventChain each reading can be its own entry. EventChain exists to give different interested parties the certainty that every one of them holds the same history, at the speed and cost of a traditional message queue. It records first and agrees

afterwards. Every holder compares its copy with a one line history hash that the Hub signs and publishes. This paper proposes that an independent foundation appoint hosts to these hashes as a neutral, trusted third party (Section 5.1). This third party comparison mechanism gives the certainty the published histories are the same for every holder. This paper calls the result consensus by comparison (Chapter 3). The mechanism is in Chapter 4.

The art-registration service ascribe recorded on its farewell page that a marketplace weighing its service registered 100,000 photos a day, as many transactions as the whole Bitcoin network then handled, and that each photo needed its own provenance entry (ascribe team, n.d.). EventChain now makes provenance at scale fast, simple and affordable. What becomes worth building is everything a blockchain made too expensive or too slow to record. You can find five worked examples with these requirements. An electricity supplier can send each fifteen-minute reading the meter's key signed. The customer recomputes the total themselves. An insurer and its policyholder can hold every estimate and revision behind a claim. A dispute then starts from evidence both already hold. An enterprise that hands confidential documents to an AI supplier can keep the supplier's signed acknowledgment of each document version and the terms of its use. A lender can keep both valuations behind a mortgage approval. A later reviewer sees which one the approval relied on. A buyer of an aircraft part can check the signatures of the maker and each handler against registered keys before fitting it. Buyers once accepted forged paper certificates for such parts (Serious Fraud Office 2026).

A valid chain shows which key signed what, and in what order. The chain cannot show whether a statement was true. The party bound to the signing key answers for that (Section 3.3). The history records the approval or the transfer. Whatever moves the money or the title acts on that evidence, in the application or blockchain built for it (Section 3.5). The place to start is the process where waiting for reconciliation costs you most. The first entry is useful the day someone signs it. Nobody else has to agree first.

2 Reconciliation After the Fact

Suppose a buyer receives fewer cartons than the carrier's invoice charges for. Each side rebuilds the delivery from its own records, and the two versions first meet at the invoice. Days go into working out which one is right. This chapter shows why that work exists, why it costs more than it did five years ago, who has kept the history so far, and what a history must have before the parties can stop rebuilding it.

2.1 Each Party Keeps Its Own Records

Today a company can check a document. The actions around it, who issued it, who relied on it, and what happened next, sit in emails and in other companies' systems. Each party keeps its own records of a shared process, and the whole history exists nowhere, so whoever needs it rebuilds it from the parts it can reach.

Even two parties who trade every day lack a history of their shared work that both hold. In 2020 FreightWaves reported that up to 70% of Walmart Canada's carrier freight invoices had "some sort of issue". Staff spent their time sorting out the charges, and carriers waited for payment. Then the retailer and its carriers worked from one shared history of each shipment. After that, the carriers disputed fewer than 2% of more than 150,000 invoices (Tabak 2020). The invoices did not disagree because anyone lied. They disagreed because each side reconciled the shipment from its own records and met the other side's version only at the invoice.

Where reconstruction falls short, organisations fill the gap with trust, as when a manufacturer trusts a supplier's database. The supplier can change a record in it later, or show different parties different copies, and nobody outside would notice.

2.2 Why Now

Four changes make a history that outsiders can check worth more than it was five years ago: more value rests on digital claims, faking evidence has become cheap, the burden of proof is moving onto the company, and the tools to sign at source already sit in people's hands.

Governments are moving transactions that once rested on paper onto digital claims. Dubai Land Department forecasts that tokenised property will reach AED 60 billion, "7% of Dubai's total real estate transactions", by 2033; the figure is the department's own forecast for a pilot launched in March 2025 (Dubai Land Department 2025). Trade documents are moving the same way. Under the UK's Electronic Trade Documents Act 2023, "An electronic trade document has the same effect as an equivalent paper trade document" (s.3(2)) (UK Parliament 2023), and 13 jurisdictions have adopted UNCITRAL's Model Law on Electronic Transferable Records, including Singapore and Abu Dhabi Global Market, several of them in part (United Nations Commission on International Trade Law 2025). A digital title or an electronic bill of lading is worth what the evidence behind it is worth: who checked the identity, who valued the asset, who approved the transfer, and in what order.

Faking the evidence has become cheap. In 2024 an employee in the Hong Kong office of the engineering firm Arup made 15 transfers totalling HK$200 million, about US$26 million, after a video call on which everyone except the employee was a deepfake (Hong Kong Free Press 2024; CNN 2024). The call carried every signal the employee normally relied on: familiar faces, familiar voices, a senior instruction. A check on who signed, with a key held in the signer's own device, still holds, because a deepfake cannot produce that signature.

The burden of proof is moving onto the company. Since 21 June 2022, US Customs and Border Protection has presumed that forced labour made any goods produced wholly or in part in Xinjiang, and the importer must rebut the presumption (U.S. Customs and Border Protection 2026). The evidence that rebuts it originates with suppliers, their suppliers, and the facilities that made each input, each keeping its own records. An importer that cannot show that history in an order customs can check carries the loss.

The signing tools are already in place. The FIDO Alliance, the industry body behind passkeys, reported 5 billion passkeys in use worldwide in May 2026 (FIDO Alliance 2026), and Windows 11 requires "Trusted Platform Module (TPM) version 2.0" (Microsoft 2026). People already carry the credential that can sign a statement, and computers already hold hardware that can protect a signing key.

2.3 Who Keeps the History

Each generation of trust infrastructure changed whom an outside party has to trust, and only the last generation hands that party the whole history (Table 2.1).

Table 2.1: Whom an outside party still trusts for a history, by generation of trust infrastructure

Generation	Capability gained	Outside party still trusts
Relationships	Action through known counterparties	The counterparty's memory and good faith
Documents	Claims travel beyond their authors	The issuer and every custodian since
Databases	Structured coordination at scale	The operator's account of its own records
Blockchains	Shared state without one controlling operator	The consortium's members, consensus, and governance
Portable signed evidence	Every interested party holds the full history and reconciles it as it happens	The signers and the enrolment evidence behind their keys; the heartbeat hosts (parties that hold the Hub's heartbeats and that the Hub does not control); the Hub's availability to accept new entries

Each earlier generation made the history easier to share, but each left the outsider holding only its own records, which it reconciles against the others' after the fact. In the last generation every interested party holds the whole history and reconciles it as it happens. The Hub, the one service that orders and distributes entries, publishes a signed statement of the newest entry at an agreed interval (a *heartbeat*). Each holder checks that every entry carries its producer's signature over everything before it, and that its copy matches the latest heartbeat (Chapter 3).

2.3.1 Why Blockchains Stalled

Several enterprise blockchain consortia closed in 2022 and 2023. Their principals cited funding, participation, and commercial viability (A.P. Moller - Maersk 2022; Wragg 2022b, 2023a; McGee 2022). Participants named what consumed the money. One, quoted in Global Trade Review, called it "a business model problem more than anything ... people underestimate the time and money required to reach critical mass" (Wragg 2022a). Swiss Re's chief executive said of B3i, a consortium of insurers of which Swiss Re was a member: "It would need all insurance companies to basically create smart contracts at the beginning, at the origin" (Howard 2022). We read this as the architecture's cost surfacing as a funding problem: a network that is useful only once most participants have joined needs financing until they do.

TradeLens reached ten ocean carriers and connectivity to about two-thirds of global containerised freight (IBM 2021), and still closed. In 2018, Hapag-Lloyd's chief executive, Rolf Habben Jansen, named the objection: "each individual project claims to offer an industry platform that they themselves control. This is self-contradictory" (Hill 2018). The UN's trade-digitalisation database records that "the shipping industry didn't have the appetite for sharing confidential data with a competitor" (United Nations ESCAP 2025; Najati 2025). Price and the industry's reliance on paper documents weighed as well; only 1.2% of bills of lading were electronic (PierNext 2023; Wragg 2023b). Neither cause belonged to the protocol, and both have moved since: electronic trade documents now carry the legal effect of paper (Section 2.2), and a history that costs one write per event has no per-container price to charge (Section 3.2).

Where one accountable operator already existed, a shared blockchain added little. ASX's independent review found "little value to processing all the business logic on-ledger as ASX maintain data integrity as the market operator", and that consensus contributed to round-trip latency; settlement runs of more than 100,000 transactions failed under load (Accenture 2022). ASX wrote off A$245–255 million (ASX Limited 2022).

The blockchain projects that lasted solved one business problem, treated the blockchain as an implementation detail, and had a party able to bring participants in: a regulatory mandate, a party with workflow authority, or an existing orchestrator (Gottlieb 2026). Where no such party exists, portable signed evidence needs no one to bring participants in, because each outside party checks the evidence it holds without joining anything.

2.4 Five Requirements

Portable signed evidence needs five properties at once.

Table 2.2: Provenance requirements and their practical meaning

Requirement	Practical Meaning	Failure Without It
Tamper-evidence	Any holder detects alteration or insertion; a check against the Hub's latest heartbeat detects truncation and divergent copies	Whoever keeps the history can change it after the fact, and others cannot detect the change
Attribution	A key signs each entry, and enrolment evidence identifies the key's owner	Responsibility diffuses across departments and service accounts

Requirement	Practical Meaning	Failure Without It
Independent verifiability	Any authorised holder can check the history without relying on the system that produced it, once the holder has the evidence each check needs	Auditors must trust the operator they are auditing
Every event, every party	The history receives every reading, revision and approval from every party, large or small, at whatever volume the work produces	High-volume sources and smaller parties fall outside the history, which then has gaps where disputes start
Continuity	One ordered history across parties and systems, in which physical checks, human decisions, and automated outputs all sit	Nobody can place a piece of evidence in sequence with what happened around it

3 Consensus by Comparison

The members of a ledger agree on the history before each append, by a round of votes among everyone present. The holders of an EventChain history agree after the append, each by comparing its copy with one line the Hub has signed and published (a *heartbeat*), and nobody has to be present. Both routes end in the same state: every party holds one history, and each knows that any other party checking against the same heartbeat finds the same history through the entry it covers. Because each holder applies the same rules to an identical copy, each reaches the same verdict on whether an entry follows them; only the payload check depends on what each holder may read (Section 4.7). That state is what consensus guarantees, and this paper uses the word for the guarantee. The two routes differ in when the agreement happens and who has to be present. The costs and limits of each route follow from that difference.

In the same way, a history with two or more holders is a distributed ledger in which the parties hold the copies and one Hub sets the order: many producers write, one Hub orders each topic, holders read, and no holder's check depends on another holder.[1] This chapter says "a ledger" for the kind that votes, and a history with one holder is a signed log.

3.1 What You Get

EventChain gives six of the eight guarantees people seek from a ledger without a round of votes, each under an assumption that Table 3.1 names, and leaves the other two to contract and to other layers.

[1] In the vocabulary of distributed systems, a topic is a single-writer replicated log (the Hub is the single appender; producers sign the entries it appends) with timeline consistency, the model of PNUTS, in which "all replicas of a given record apply all updates to the record in the same order" and a read returns a possibly stale version that "is always a valid one from the record's history" (Cooper et al. 2008). Under PACELC (Abadi 2012), a topic chooses consistency over availability when partitioned, refusing to append against an unknown head or to act on evidence it cannot confirm, and latency over consistency otherwise, appending without waiting for holders; the heartbeat bounds undetected staleness to one publication interval while a host the foundation lists for the topic stays reachable and preserves every heartbeat (Section 5.1). Abadi classifies PNUTS the same way, PC/EL.

Table 3.1: Ledger guarantees and where EventChain gets them

Guarantee	A ledger gets it from	EventChain gets it from	What remains assumed
One history that every party holds and agrees on through the latest committed or published entry	A round of votes before each append	Each holder compares its copy with the latest Hub-signed heartbeat (Figure 4.2)	The hosts the foundation lists for the topic preserve every heartbeat and serve everyone the same list (Section 5.1)
Nobody can rewrite an entry unnoticed	Every replica validating every block again	Each producer signing its entry over the hash of the entry before it (Section 4.1)	Holders run the verifier
Events keep their order	The same round	One Hub thread appending in sequence, each signature fixing its entry's position (Section 4.5)	The Hub decides which of two competing submissions goes first
Who made each statement	Organisation certificates or wallet addresses	A key per signer: a person's passkey, a device's hardware key or a service's key (Table 4.2)	The enrolment evidence that binds the key to a party, and the registry root key that the parties accept outside the protocol (Section 4.3)
Verification without the operator	Running a node	Running the open verifier on the file, offline (Section 4.9)	A correct verifier, whose source anyone can read
Time	Block timestamps	The Hub's asserted time, and an optional anchor in Bitcoin (Section 4.8)	The Hub's clock; for the anchor, the Bitcoin network. Neither dates the event itself

Guarantee	A ledger gets it from	EventChain gets it from	What remains assumed
Nobody can block a writer	Open membership and many validators	Not supplied: a Hub can refuse an append; a producer retains signed evidence of its attempted submission (adversarial Hub analysis)	Hub availability and acceptance remain contractual duties
Settlement among strangers	A round of votes on balances	Not supplied: ownership and balances belong to an application or ledger (Section 3.5)	Each party answers for what it signs

The history is a text file that anyone can check with SHA-256 and ECDSA-P256, so the evidence outlives the platform that produced it.[2] Every holder verifies every entry of its history, including entries whose payloads access policy keeps from it (Section 4.7). A gap in the chain or a copy shorter than the latest heartbeat shows that entries are missing, so a holder notices when a history leaves out inconvenient entries.[3]

3.2 What It Costs

Because nobody has to be present at the append, an append is one write, and the work of recording grows with the number of events and with nothing else. Picture an electricity supplier with 10,000 meters, each signing a reading every fifteen minutes (worked example): 960,000 entries a day, about eleven a second. Whether the supplier can keep that history depends on whether the work grows with the number of events or with the number of parties who hold the history. On a ledger it grows with both.

In 2018, Hyperledger Fabric's authors at IBM Research reported more than 3,500 transactions a second for a payment workload with five peers in one data centre, falling to between 1,400 and 2,000 with 100 peers across five data centres, every peer holding

[2] Hash-linked time-stamping showed that ordering and tamper-evidence come from chained commitments with no round of votes (Haber and Stornetta 1991). The standards comparison places them alongside earlier systems.

[3] Parakeet frames a related pattern: public commitments, private querying and verifiable proof (Malvai et al. 2023).

the whole ledger (Androulaki et al. 2018). The whole Ethereum network processed 3.6 million transactions on 28 April 2026, its busiest day up to October 2026 (Etherscan 2026); this one supplier's readings would equal a quarter of that day. A consortium ledger recovers its costs from its users: in 2021, TradeLens listed its service from $8 per container (TradeLens 2021).

Suppose another logistics company joins a shipment process. Figure 3.1 follows the carrier's signed acceptance: the company joins the consortium ledger as a validating member, or receives the EventChain history as a holder.

A ledger can also serve read-only users without adding validators. Such a user trusts the validators for what an EventChain holder checks itself from its copy. Adding an EventChain holder adds one copy of the file and that holder's own checks; existing holders do not repeat verification for it. The application's business rules still apply: if dispatch requires the new company's acceptance, the dispatch system waits for it (Section 4.7).

The Hub checks each submitted line as a verifier does, for well-formedness, predecessor hash and signature, before it appends (Section 4.4). A benchmark of the reference Hub on 14 October 2026, on one thread per topic, committed more than 11,000 entries a second with all three checks on every line, and 29,000 with the head comparison alone, from one run (EventChain 2026b); the checks cost the Hub thread 65 microseconds a line (Section 4.5). No fee attaches to an event. Each holder receives one line of about 335 bytes per event and verifies its copy on its own processor at its own pace. A holder that verified yesterday's copy checks only today's new entries. These measurements describe different workloads and environments: observed Ethereum traffic is not network capacity, and the figures are not a comparative throughput benchmark.

Batching spreads the trip to the Hub across many events. A benchmark of the reference producer on 13 October 2026 measured it against a compare-and-append Hub model (EventChain 2026d). Each producer signed a sequence of entries, each over the one before, and submitted them together. The Hub model accepted a batch when its first entry followed the current head, and the reference Hub checks every line before writing. That first check binds the batch to the whole history. The benchmark then checked format and signatures on every output file, and all 160 passed. Table 3.2 shows the committed entries per second with a 20-millisecond in-flight delay; the measured delay was about 21 milliseconds.

Table 3.2: Entries committed per second into one topic at a 20-millisecond in-flight delay; ranges across two four-second runs at saturating load

Entries per batch	One producer	Ten producers	Twelve producers
1	46	48	48–49
100	3,873–3,898	3,999–4,024	4,025–4,048

Entries per batch	One producer	Ten producers	Twelve producers
300	9,071–9,074	8,772–8,924	8,471–8,621
1,000	17,241–17,247	12,748–13,743	11,497–12,244
3,000	23,239–23,243	14,245–14,246	12,743–12,745

Each event keeps its own signature and its own line in the file. Larger batches spread the delay across more entries, so throughput approaches one producer's signing rate. Producers that race for the same head spend more CPU: under saturating load, ten producers make about ten signatures per committed entry, because each loser signs its batch again against the new head. Larger batches also make producers wait longer. With ten producers and batches of 3,000 at this delay, median latency from the first signature to acceptance was 1.02 seconds, and the 99th percentile was 3.29 seconds; an event also waits for its batch to fill. Some producers committed nothing during the four-second windows, so a process that needs independent producers to progress gives them topics of their own (Section 4.5).

Table 3.3 applies these figures to the supplier.

Table 3.3: Fleet arithmetic: 10,000 meters, one reading every fifteen minutes (960,000 entries a day, about eleven a second)

Quantity	Figure
Hub-thread utilisation at 65 microseconds a line	About one part in 1,400
File growth per holder	About 320 MB a day
Verification time per holder, on one core	About 40 seconds per day of entries; October 2026 benchmark of 42 seconds per million entries (EventChain 2026c)

The same arithmetic covers every photograph behind an insurance claim (worked example), because the file carries a fingerprint of each photograph and never the photograph itself (Section 4.2). The cost that grows with volume is each holder's storage, about 117 GB a year at this supplier's rate.

Some work costs the same with a ledger or with EventChain: a deployment still pays to integrate existing systems and to enrol signers. Ledger consortia carried one more item: those that closed in 2022 and 2023 named reaching critical mass as what consumed the money (Section 2.3.1). EventChain has no such item, because one organisation can start and the others take the evidence when they need it.

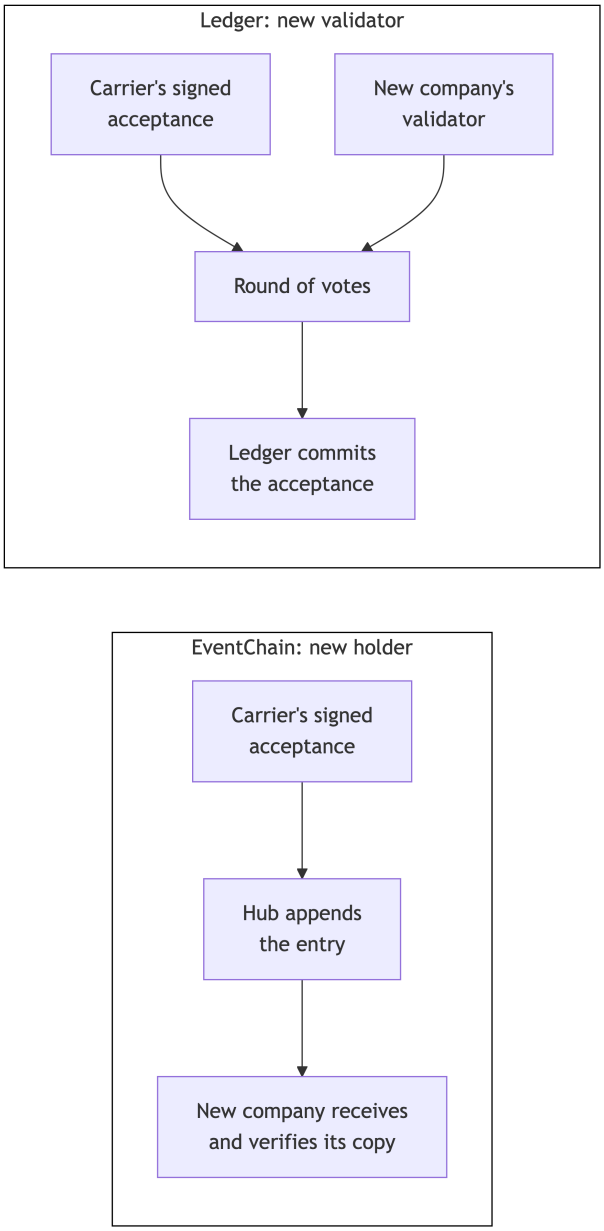


Figure 3.1: A new validator participates before ledger commitment; a new EventChain holder receives and checks its copy after append. The arrows show dependencies, not elapsed time.

3.3 What Remains Trusted

Beyond the file and a verifier, a holder depends only on what the last column of Table 3.1 names, chiefly the enrolment evidence behind each key, the heartbeat hosts the foundation lists for its topic and the Hub's availability to accept new entries. The Hub cannot rewrite a signed entry. It can withhold entries from a holder or show different holders different histories; Figure 4.2 shows how a holder checks against the latest heartbeat. Two conflicting heartbeats under the Hub's key attribute the fork to the Hub. The field calls these properties fork consistency and accountability (Section 4.4; adversarial Hub analysis). Deciding what to do with the evidence stays with the application. EventChain does not prevent a false entry at the moment of creation; it makes later denial and editing unattractive, because each entry names a signing key and binds its statement to the preceding history.[4] Table 6.9 gives the boundary capability by capability.

3.4 Where It Breaks

A valid signature proves that a key signed these bytes. It cannot reveal an honest mistake, coercion, collusion or a signer who signs without reading. Process controls and audit remain necessary, and when something goes wrong they now start from a signed history and the identity evidence behind it (failure-mode tables).

3.5 When It Fits

EventChain fits where each party to a process answers for what it signs, and where at least one of them needs evidence that survives beyond any single platform. Because each party answers for what it signs, detection after the fact suffices. Because the evidence travels as a file, nobody has to join a network before the first entry is useful. Where strangers settle payments with one another, a ledger's round of votes remains the right tool. Where one organisation controls the whole process and nobody outside needs to verify, an ordinary audit trail does. Table 6.14 sets each approach beside the others.

[4] The US Food and Drug Administration's data-integrity guidance rejects shared logins for the same reason: "a unique individual cannot be identified through the login" (U.S. Food and Drug Administration 2018).

4 How EventChain Works

Producers sign entries into an append-only file (AOF), a single Hub orders and distributes the file, and every holder verifies its own copy. A *topic* is one history (the events a deployment collects together in one AOF). Checking the chain and the producer signatures needs only the file and no trust in the Hub; naming the signer and confirming that a copy is current need further evidence (Section 4.9.1). The byte-level rules an implementer needs are in the Specification appendix.

4.1 Append-Only File

The AOF is a JSON Lines (JSON Lines 2023) document, one JSON object per line. Each entry carries a salted hash of its payload, a locator for the payload, the SHA-256 hash of the previous line, the signer's public key and an ECDSA-P256 signature over every other member (Table 4.1, Figure 4.1). Producers encrypt payloads before they reach the Hub, so the Hub holds or relays ciphertext only and never has plaintext or decryption keys.

Table 4.1: Base-entry fields (revision-specific members are in the entry specification)

Field	Example	Purpose
`entry_id`	`evt-042`	Producer's reference; arbitrary text that need not be unique
`payload_hash`	`obPN0cqZ...`	Salted hash of the plaintext payload (Section 4.2)
`payload_ref`	`ref-042`	Opaque locator for the payload
`prev_hash`	`_-bFxIQ3...`	SHA-256 of previous line's bytes
`public_key`	`Aw4rK2p1...`	Signer's public key
`signature`	`8QDxLmE7...`	ECDSA-P256 signature over `entry.data()` (entry specification)

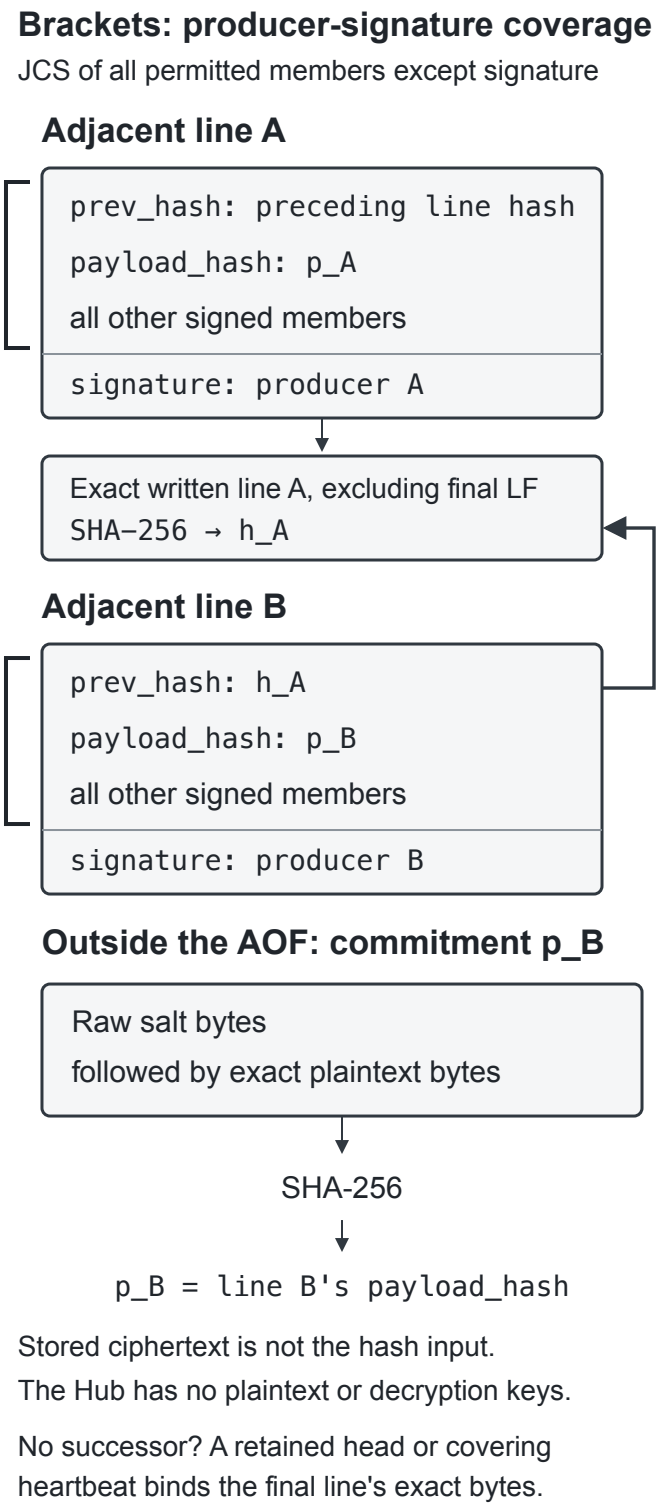


Figure 4.1: The producer signs the predecessor hash as well as the payload commitment. Moving line B to another predecessor changes its signed input. Hash links alone do not authenticate the chain.

Because the signature covers `prev_hash`, each producer signs its own position in the chain. Editing any line or moving an entry to a different predecessor invalidates a producer signature that the Hub cannot recreate, so the ordering guarantee rests on the producers' keys. An entry's unique identifier, its *Event ID*, is the SHA-256 hash of its exact written line, and later entries refer to earlier ones by it.

4.2 Payload Hash

`payload_hash` is `SHA-256(salt || payload)` over a fresh random salt followed by the exact plaintext bytes. Nobody without the salt can test guesses against the hash, and whoever holds the plaintext and the salt can recompute it, so each holder keeps the salt with the payload. The AOF names no signer, only a public key; the signer's identity and role sit in the payload under access policy (Section 4.7).

4.3 What a Signature Proves

A device or a software service signs its entries directly: a sensor signs each measurement at source with a hardware-held key. A person's passkey cannot sign arbitrary bytes, so the producing system signs the entry and the person confirms it in a separate *Mint* entry whose WebAuthn challenge is the confirmed entry's Event ID (WebAuthn checks). In every case a valid signature proves that a key signed these bytes. What it proves about who acted depends on where the key lives (Table 4.2).

Table 4.2: What a valid signature proves for each kind of signer

Signer	Where the key lives	What a valid signature proves
Device	TPM, physically unclonable function (PUF) or secure element	This key signed these bytes. Binding the key to a particular device rests on enrolment or attestation evidence and on the key's actual protection; the signature shows nothing about calibration, attachment or the measurement path

Signer	Where the key lives	What a valid signature proves
Software service	Cloud key-management service	Something able to use the key signed. The file does not say which workload, person or administrator caused the signature
Software in an attested enclave	Enclave bound to measured code	Under the platform's attestation assumptions, the key belongs to the measured code. A model counts only if the validated measurement covers it
Person	Passkey	The passkey answered this challenge with the checked flags. Naming the person and role rests on enrolment evidence and passkey-control policy

A *registry* binds each public key to a person, device, service or role, with the enrolment evidence behind the binding. A registry is a topic like any other, and a holder verifies it with the same checks. It cannot write, reorder or block an entry in any other topic. A binding that others rely on carries two signatures, the subject's offer and the registry keeper's acceptance, so neither party creates it alone (registry conventions, three of them still open). An organisation keeps a registry for its own signers. An authority or community keeps one that binds each organisation's root key to its legal identity.

Each topic declares the registries it accepts, by root key directly or through a binding in a registry it already accepts. It changes that declaration only with its other parties' countersignature, because otherwise the party that started the topic could swap registries and re-attribute past signatures. The chain of registries ends at a root key that the parties accept outside the protocol, as a browser accepts its root store (Reddy and Wallace 2010). A binding shows whom its keeper says a key belongs to.

A signature carries no ongoing authority and no time, so a departed employee's earlier signatures stay valid. A lost key leaves the chain intact: the keeper retires it and the organisation enrols a new one. A compromised key differs, because someone else may sign with it. The keeper therefore signs a compromise claim that names the time from which the key is suspect, and the claim stands beside the history as a visible dispute that erases nothing. The verifier reports each entry the key signed between that time and the claim as inside the claimed window, and the application decides. A payload may

state a time, but the signer asserted it, and whoever holds a stolen key asserts whatever suits them. So the verifier places an entry in time by the heartbeats and anchors around it (Section 4.4.1, Section 4.8), and no finer than they allow. The verifier reports signature validity, binding evidence, compromise status and authorisation separately.

4.4 Append Protocol

The *head* is the hash of the last line in the chain. A producer keeps the head it last appended or observed (its *retained head*) and submits against a verified *expected head*. Because the signature covers `prev_hash`, changing that field requires signing again.

1. The producer requests the current head and any entries since its retained head. It accepts a matching head with no new entries, or verifies an extension to the new head. If the Hub supplies a history that omits the retained head, the producer treats it as evidence of rollback or truncation and does not sign against it.
2. The producer prepares a *submission* of one or more entries. It sets the first entry's `prev_hash` to the verified current head and each later entry's `prev_hash` to the hash of the entry before it in the submission, signs each entry, and submits them together with that expected head (submission format).
3. The Hub accepts the submission only if its current head still matches the expected head, and appends it whole or not at all. After the append, the producer retains the new head.
4. If another submission arrived first, the Hub rejects this one. The producer obtains and verifies the new extension, sets the first entry's `prev_hash` to the new head, signs the whole submission again and resubmits against that head.

The Hub applies to each line of a submission exactly the checks a verifier applies to one line: the line is well formed under the entry specification, `prev_hash` equals the current head for the first line and the hash of the preceding line for every later one, and the key on the line signed it. It appends every submission whose lines all pass, whole or not at all, whatever the entry says and whoever holds the key, and judges nothing else; a false statement or an unbound key enters the history, where the verifier reports it and the other parties contest it (Section 4.6). A submission that passes receives a signed receipt, the Hub's commitment that it is in the history at that position, and one that fails receives a signed refusal naming the submission hash, the line that failed, the reason and the Hub's asserted time (receipt and refusal). The submitter retains it, so a refusal of a valid submission is evidence against the Hub.

A valid chain can still be an old prefix, so the protocol requires the Hub to issue signed *heartbeats* that trusted *heartbeat hosts* pull and publish. Every holder, including a first-time one, retrieves the latest heartbeat for its topic from a host and checks its copy against it, and so detects an out-of-date copy or a diverging history without contacting

other holders. Two conflicting heartbeats under the Hub's key attribute the fork to the Hub. These are the established properties of fork consistency (Li et al. 2004) and accountability (Laurie et al. 2013). The publication interval bounds freshness, and each host must preserve every heartbeat and show everyone the same list.

4.4.1 Hub Heartbeat

A heartbeat carries the topic's head, the entry count and the time of issue under the Hub's signature. The Hub issues one at the agreed interval even when no entries arrive and serves it at its heartbeat endpoint. A heartbeat host pulls it at that interval, checks the signature under the accepted Hub key and, holding the file, that it names the file's head, then publishes it. A host preserves every heartbeat for each topic it serves and publishes the Hub key the topic accepts; a Hub that stops issuing heartbeats shows as a gap at every host. Section 5.1 proposes who should decide which hosts a topic trusts. Every holder compares its own copy with the heartbeat a host serves (Figure 4.2). The time is the Hub's own assertion; independent timestamping can add external time evidence (Section 4.8). Heartbeat Encoding and Retrieval specifies the format and retrieval.

Suppose the Hub serves different histories to a buyer and a warehouse that share one topic. Both copies pass their chain and producer-signature checks, but they contain different hashes at the same position. The warehouse detects the disagreement by comparing its hash with the heartbeat the host serves (Figure 4.2), without contacting the buyer.

The comparison needs a recent heartbeat under the accepted Hub key, from a host that serves everyone the same list. A heartbeat vouches for the history up to the entry it names and for nothing after it. A copy that stops short of that entry is missing part of the published history, and an old heartbeat says nothing about whether a copy is current. A matching heartbeat shows that a holder has the same history as everyone else who checks it. Whether that history is genuine is a separate question, which the producer signatures answer: a Hub that appended a forged entry would publish a heartbeat over it, and only the signature check on that entry catches it.

4.5 Scale and Ordering

The Hub appends on a single thread per topic, so each topic has one ordered history and different topics proceed independently. Each accepted submission costs one append and no multi-party consensus round. A holder that keeps a verified prefix verifies only the extension; Section 3.2 gives the measured rates.

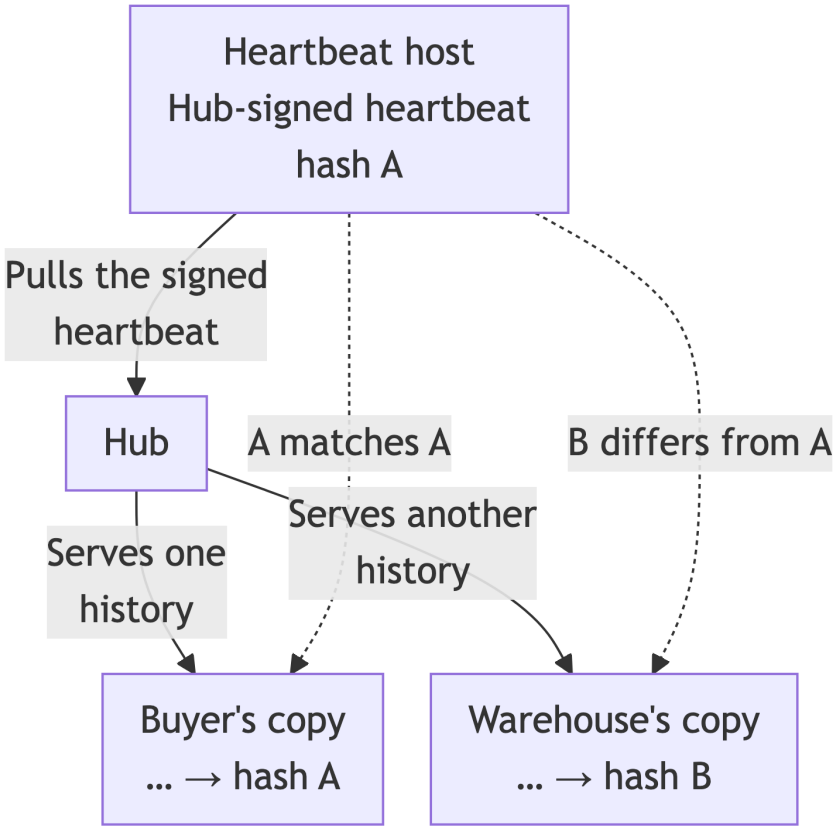


Figure 4.2: Different hashes at the same position cannot both match one published heartbeat. A and B stand for those hashes in the same topic. A conflicting copy alone does not attribute a fork to the Hub; that requires conflicting Hub-signed heartbeats.

4.6 Cryptographically Bound Contracts

A Cryptographically Bound Contract (CBC) records a two-party agreement as two entries: the offering party signs an offer, and the accepting party signs an acceptance that names the offer by its Event ID (CBC references). A custody transfer is one instance; a supplier's acceptance of shared data under agreed terms is another (worked example). Until the acceptance exists the CBC is *pending*, and any party inspecting the chain sees the missing acceptance.

A correction, an objection or a resolution is likewise a new signed entry that names the entry it addresses (correction references). The original stays in the chain, no later statement overrides it automatically, and application policy decides who may make each kind of statement.

4.7 Access to Payloads

Applications decide which holders may retrieve and decrypt each payload, and a Hub or storage service can enforce that without holding decryption keys. A regulator may read every payload while a customer reads only those under its contract, yet both hold the same AOF and run the same checks on it (payload access and export). An application the parties authorise can act on the evidence as soon as it meets the application's rules, with no person checking each step again. Systems subscribe to a topic over HTTP and receive each entry as the Hub appends it. Each keeps its own copy and verifies it.

4.8 OpenTimestamps

An optional daily OpenTimestamps (OTS) rollup anchors the current chain head in Bitcoin, which shows that the committed bytes existed before the time of a validated anchor (OpenTimestamps 2026; Todd 2016). That evidence relies on the Bitcoin network and on validation of the proof, independent of the Hub, and does not show when the events occurred. A Hub can anchor both branches of a fork, so OTS protects against neither split views nor stale copies.

4.9 Verification

The verifier reports separate conclusions about a history (Table 4.3). A holder can check its chain and signatures without payload access, and can validate an external time proof without the plaintext.

4.9.1 What Each Check Needs

Table 4.3: Evidence each verification check needs and what it establishes

Independent check	Evidence beside the AOF	Check and result
Chain and producer signatures	None	Check predecessor hashes and signatures: each entry binds to its predecessor and each signature verifies under the key embedded in it. Hash links alone authenticate nothing.
Named attribution	Enrolment and compromise/dispute evidence	Report key validity, identity binding and dispute status separately (Table 4.2).
Freshness and shared history	Accepted Hub key, publication settings, authenticated heartbeat from a named host	Compare the covered-position hash; report match or deficiency (Figure 4.2). The heartbeat covers only that prefix.
Payload commitment	Exact plaintext and retained salt	Hash raw salt bytes followed by exact plaintext bytes. A match shows that these are the bytes the entry committed to.
External time (optional)	Complete OTS proof and validation context; no plaintext	Validate proof and coverage: existence before anchor time, not event time (Section 4.8).

The application decides whether the evidence suffices and whether the signer had authority to act. If heartbeat evidence is missing, stale or conflicting, it pauses dependent actions for the topic, keeps capturing and queuing events locally, and escalates (heartbeat outcomes).

5 Open Protocol

Holders can verify an EventChain history without access to the application that created it. They retain the file and use the verifier to check its chain and signatures, even after changing suppliers or closing an old system. Identity, payload, and time checks also need their supporting evidence (Section 4.9.1).

The file uses JSON Lines (JSON Lines 2023), SHA-256 and ECDSA-P256 (National Institute of Standards and Technology 2015, 2023), JCS (Rundgren et al. 2020) and WebAuthn (Hodges et al. 2021), so a verifier built from this paper confirms any AOF, past or present, whichever Hub produced it.

The public EventChain implementation repository contains the Haskell producer, a separate verifier package, and shared cryptography code under the BSD-3-Clause licence (EventChain 2026a). The reference Hub lives in its own repository (EventChain Protocol 2026).

The holder's checks are independent, each needing the evidence named in Table 4.3. Verification results inform an application decision; they do not authorise it.

5.1 Protocol Stewardship

Every check in this paper runs on one copy of the history. Each entry carries its producer's signature, so the Hub cannot rewrite history. One power remains with the Hub: it can show one holder one history and another holder another, and each copy passes every check on its own.

The file alone already gives fork consistency: holders who compare copies with one another find the fork. That comparison is a distributed system, and EventChain exists so that no holder has to coordinate with another. A host that keeps the Hub's heartbeats strengthens the guarantee to the point where one holder, on its own, detects a fork or an old copy within one publication interval.

At each interval the Hub signs a heartbeat, and the hosts the foundation lists for the topic pull it and publish it (Section 4.4.1). Figure 4.2 shows how a holder compares its copy against the host's latest heartbeat, including a copy that ends early or diverges at the covered position. Two conflicting Hub-signed heartbeats for the same topic and

position attribute a fork to the Hub; a mismatching copy alone does not. Publication is interval-based, so the current heartbeat covers the prefix up to its position, and entries after it await the next heartbeat.

A heartbeat host is the trusted party in the design, and the design asks little of that trust. It cannot forge a heartbeat, because the Hub signed it. It cannot read a payload, because the file carries only hashes. It can lie only by showing an old list, which a holder notices when no heartbeat arrives within the agreed interval, or by showing two holders two lists, which any two holders expose by setting their heartbeats side by side. A topic with two hosts lets one holder make that comparison alone. Haber and Stornetta's company printed its weekly hash as a classified advertisement in the New York Times for the same two properties: nobody can withdraw a printed line, and every reader sees the same page.

Who may host heartbeats is the one thing in the design that has to be neutral, so we propose that an independent foundation with no owners keep a signed list of the hosts it trusts for each topic. The foundation also keeps what every party already shares: the specification, the open-source verifier, the conformance suite, the registry conventions, and a signed list of recognised Hub service providers (the parties that run Hubs) for topics that span rivals. A founding authority convenes the foundation and controls neither its council nor its oversight; other authorities join that oversight as equals, and no supplier or founder decides alone.

A signature names a key. An entry is evidence only if someone stands behind that key and nobody can change later who that was. A *registry* settles the first: a topic, kept by any organisation or authority, in which the key owner and the registry keeper both sign each binding of a key to a person, device or role, so neither binds a key alone. The protocol settles the second: a registry cannot touch an entry in any other topic, and a topic changes which registries it accepts only when its other parties countersign, so nobody changes who signed a past entry by changing which registry answers. Every chain of registries ends at a root key that the parties accept outside the protocol, as a browser accepts its root store (Section 4.3). The foundation keeps the conventions those registries follow; binding a key stays the keeper's act, and accepting a root stays the parties'. Who may keep a registry that others accept, and how a keeper recovers a lost root, are the first questions the foundation settles (registry conventions).

5.2 Bootstrapping

One organisation starts on its own. It enrols the keys its people and devices already hold, runs a Hub for one process it already operates, and signs that process's events into a topic. Its Hub publishes heartbeats from the first entry (Section 4.4.1). Each counterparty takes a copy of the file when it needs one, verifies it against the latest heartbeat, and so

nobody waits for a consortium. The deployment tables list what a production deployment still has to put in place.

6 Conclusion

A process on EventChain leaves signed evidence of every reading, estimate, approval and handoff as it happens, and everyone with a stake in it holds the same history. Reconstruction after a shipment or a claim has no work left. A dispute opens from evidence both sides already hold.

Software can act on evidence it already holds. An approval that today waits for a person to check what the other side sent can follow the moment the evidence meets the rules the parties set, with the content still encrypted and readable only by those they allow (Section 4.7). That is what people hoped a blockchain's programmable contracts would do. An audit becomes a read of a history instead of a reconstruction from both sides' files. An asset carries its history to its next owner across companies and borders. The buyer checks it before paying. A tokenised title or an electronic bill of lading rests on evidence any recipient can check (Section 2.2). A regulator holds the history itself, with no report standing between. Every process in which you wait for someone else to confirm what happened is a candidate, and the ideas a blockchain priced out of reach are the first to look at (the worked examples).

None of this waits for an industry to agree. The specification and the verifier are open. Any party can run a Hub for its own topics (Chapter 5). One company can start with one process, and the others take a copy when they need it. We propose an independent foundation with no owners to keep the one line every holder checks against (Section 5.1). The first history can begin this quarter, in one process, with the keys your people already carry.

Appendix: Worked Examples

Every example below is a process whose parties reconcile after the fact today, and in each one the dispute or audit instead starts from a history every party already holds. The examples share one arrangement. Keys held by people, devices, and software sign the entries, and the file carries only a hash of each document (Section 4.2). Each party that may later need to prove something keeps the documents it has a right to hold and the enrolment evidence for each signing key (Section 4.9.1).

Each entry can leave the system that collected it. A party who arrives later, such as an auditor, checks the history with an open verifier (Chapter 5) and does not need the application that produced it. Because every step enters the history, it keeps details whose importance nobody saw at the time.

Every Reading Behind the Bill

Suppose a commercial customer disputes one quarter of its electricity bill. Its supplier answers with the history of every fifteen-minute reading behind the total. Each reading carries a signature from a key associated with the meter, and the customer checks the disputed readings, without access to the supplier's billing system. A supplier can send that history with every bill and make checkable measurements part of its service.

The supplier's signed report identifies the exact reading entries it used. With authorised access, the customer retains the measurement contents alongside the history. A later assessor can check those readings against their signed entries and the report, without asking the supplier to reconstruct the evidence.

That evidence lets the parties locate a disputed reading, but its signature does not establish that the measurement was correct. The parties examine the evidence that links the signing key to the meter. Calibration records and physical inspection help assess whether the meter measured correctly at the right point.[1]

[1] Tying the key to the installed meter takes enrolment or attestation evidence that the parties retain, and commissioning evidence that ties the meter to its measurement point.

Every Estimate Behind a Claim

Picture a policyholder who disputes a car-repair settlement lower than the repairer's estimate. The insurer and the policyholder hold the same history of the claim: the photographs the policyholder sent, the repairer's first estimate, the revised estimate after inspection, the assessment that recommended the settlement, and the adjuster's approval. Both sides see which estimate the recommendation used, so the dispute starts from the evidence. An insurer can build that history for every claim as the claim develops.[2]

The recommendation names the revised estimate it used, and the approval names that recommendation; entry order alone does not show which evidence a decision used. A later correction adds a signed entry and keeps the earlier decision's basis in view.[3]

A reviewer investigating a discrepancy can trace it to the contribution that introduced it, and the signature on that contribution shows which key signed it, so the party bound to that key answers for it.

Every Document Handed to an AI Supplier

Imagine an enterprise that sends confidential design documents to an AI supplier for an automated review. The contract lets the supplier analyse the documents for the enterprise, and forbids it to train shared models on them or to disclose them to other customers. On receipt, the supplier signs an acknowledgment of the exact document version and those terms, and the enterprise keeps it. An enterprise can collect that acknowledgment for every version and permission change, and keep the evidence of what each supplier accepted after the relationship ends.

The concern is common: in Cisco's 2024 Data Privacy Benchmark Study, 68% of 2,600 privacy and security professionals cited disclosure to the public or to competitors as a concern about generative AI (Cisco 2024).

The acknowledgment completes a two-party agreement, which Section 4.6 calls a Cryptographically Bound Contract (CBC): the enterprise's entry offers the documents under the terms, and the supplier's signed entry accepts them. The enterprise's own entry shows only what the enterprise sent; the supplier's signature makes the receipt

[2] Insurance fraud is the wider context: in 2022 the Coalition Against Insurance Fraud, funded by insurers and consumer groups, estimated annual insurance fraud in the United States at $308.6 billion across all lines (Coalition Against Insurance Fraud 2022). The coalition builds the total from earlier estimates and says the actual cost is impossible to measure accurately.

[3] A software signature shows that the assessment service's key signed the recommendation; showing which code or model produced the recommendation takes further deployment or attestation evidence.

evidence of what the supplier accepted.[4] Each acceptance references the exact offer, even if other entries fall between them in the history.

If another customer's output later reproduces a distinctive confidential passage, the enterprise's reviewer checks the retained documents against their hashes and compares the passage with that output and its full prompting context. Researchers extracted training data from the production language models they studied (Nasr et al. 2023). The contract's dispute process decides whether the supplier breached it, and the enterprise brings the evidence it already holds: the supplier's acknowledgment of the particular material and the restrictions it accepted.

Every Check Behind a Mortgage

Consider a lender that approves a mortgage after the valuer revises the valuation, because a new search found a right of way across the land. Years later, at refinancing, a reviewer asks which valuation the approval relied on. The history answers: both valuations remain as signed entries, and the approval references the revised one. A buyer or lender can keep that evidence for every property decision as the work happens, including the checks and revisions that never appear in the final approval.

Four contributors build the evidence behind an approval like this one (Table 6.1).[5]

Table 6.1: Evidence a property approval can retain

Contributor	Evidence retained
Identity checker	Signed checks, result and references to identity evidence
Title reviewer	Signed search findings, source, date and unresolved issues
Valuer	Signed valuation versions, assumptions and supporting material
Approver	Signed decision referencing the evidence relied on

[4] An automated receipt shows that the signing key signed the acknowledgment. Linking that key to the supplier and its authority takes further evidence. The signature does not show which model or code handled the documents.

[5] Identity checkers already keep evidence of their checks. In England and Wales, HM Land Registry contacts conveyancers and other checkers to check that a form in their name is genuine, and its guidance says it is important to keep a record of anyone whose identity they verified (HM Land Registry 2026).

Every Handoff of a Part

Between January 2019 and July 2023, the UK parts trader AOG Technics sold more than 60,000 aircraft engine parts with forged Authorised Release Certificates, the documents that guarantee airworthiness. Its director doctored genuine certificates, and customers received documents carrying the signatures of sales and quality managers who did not exist. The fraud left planes grounded in the UK and worldwide, with estimated losses of more than £39.3 million for airlines, and came to light after an airline asked the aircraft equipment manufacturer Safran to check the authenticity of one part (Serious Fraud Office 2026). If the maker or maintenance organisation signs each release and handoff of a part, and the buyer can link that key to the organisation through an accepted key registry (Section 4.3), the buyer can check the chain before fitting the part. A doctored certificate then fails the signature check, and a manager who does not exist has no key bound to them.

The same evidence serves any asset owner who inherits equipment: the inspection findings, corrections, and handoffs that people sign as routine work proceeds.

For a compressor, an inspector's finding and the fabricator's correction remain in the history together, and a later approval references the evidence it accepted without removing the earlier finding, as in the claim example. Each custody transfer leaves an offer and a signed acceptance, as in the AI example; the pair shows what the parties accepted and does not by itself prove physical delivery or equipment condition. The next owner inherits the history and authorised access to the supporting material. Before the purchase closes it checks its copy against the latest heartbeat (Section 4.4.1), and after a failure it traces which findings and approvals preceded it.

A digital product passport (a product record shared with parties such as repairers and recyclers) can present selected information from the same history, so a repairer can examine maintenance evidence or a recycler retrieve the material information it may see. The EU's Ecodesign for Sustainable Products Regulation provides for passport information shared under access rights that differ by product group (European Parliament and Council 2024). The passport application decides who may see what and whether it meets the applicable requirements; EventChain supplies the signed history beneath it, so later checks need not depend on the original passport portal.

Appendix: Specification

This appendix holds the byte-level rules that Chapter 4 relies on.

Entry Specification

Lines and rejection. Each line ends in `0x0a`, and a file that contains `0x0d` anywhere counts as malformed. A verifier that finds a malformed line under any rule in this appendix rejects the whole AOF and reports the line.

Field encodings. Every member of the entry object is a JSON string. Binary members use base64url without padding (Josefsson 2006, sec. 5):

- `payload_hash`, `prev_hash`: the 32-byte SHA-256 digest.
- `public_key`: the 33-byte SEC1 compressed point (Standards for Efficient Cryptography Group 2009, sec. 2.3.3).
- `signature`: the 64-byte concatenation `r || s`, each a 32-byte big-endian unsigned integer.

`entry_id` and `payload_ref` are opaque strings with no encoding requirement.

Event ID. The Event ID (Section 4.1) excludes the row's terminating `0x0a`. After the append, the Hub returns the new position and head hash and the number of lines written (receipt); for a one-line submission the head hash is that line's Event ID. The Hub may record Event IDs in a separate lookup index. An application keeps `evidence(row key, Event ID, salt)` beside its existing data. The Hub answers lookups, and a holder checks each answer against the Event ID and its own copy of the chain.

`entry.data()`. The signed message `entry.data()` is the JSON Canonicalization Scheme (Rundgren et al. 2020) serialisation of the entry object without its `signature` member. The removal applies to exactly the member named `signature`; the signature covers every other member the declared revision permits. Because RFC 8785 orders members by name, producers may write members in any order on the line. An unknown member or an unsupported revision makes the entry malformed.

Chain bytes. `prev_hash` is the SHA-256 digest of the previous line exactly as written: every byte after the preceding line terminator (or the start of the file), up to and excluding

the previous line's own `0x0a`. A verifier applies no canonicalisation, so reformatting a line into a form a JSON parser would read as equivalent changes its hash and breaks the next entry's predecessor check. The final line has no successor; a retained head or a heartbeat at or beyond it binds its exact bytes.

Figure 6.1 separates the two byte paths; the rules above define them.

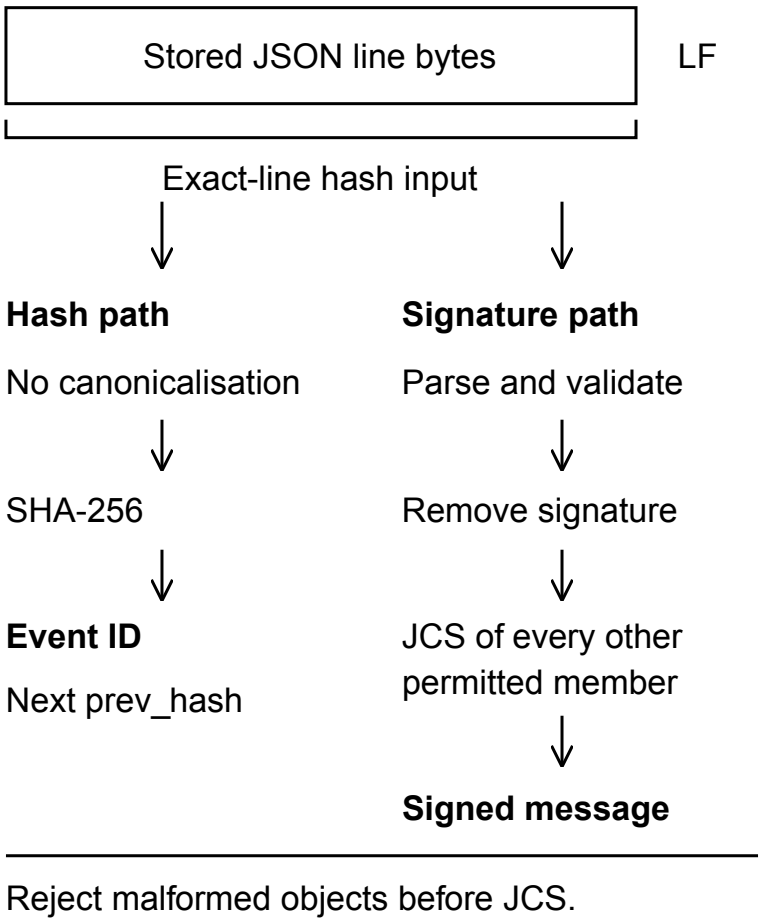


Figure 6.1: The line hash uses exact written bytes before LF. The signed message uses the validated object without `signature`, serialised with JCS. These inputs are not interchangeable.

Genesis. The first entry carries a `prev_hash` of 32 zero bytes, encoded like any other hash, and a verifier rejects an AOF whose first entry carries any other value.

Duplicate members. An entry in which any member name appears more than once counts as malformed. Parsers resolve duplicate names differently, so relying on a parser's default would let one line verify two ways.

Unicode. An escape that denotes an unpaired UTF-16 surrogate in a member name or value counts as malformed; a verifier never substitutes a replacement character.

Revisions and Mint entries. The string member `v` declares the revision. Without it, an entry has the six base members; `"1"` also admits the Mint members in Table 6.2, which record a person's confirmation of an earlier entry by passkey (Section 4.3). A verifier compares revision labels as strings, never as numbers. A revision-1 entry without Mint members is an ordinary producer-signed entry.

Table 6.2: Mint envelope for revision 1

Mint member	Value or encoding
`kind`	`"mint"`
`target_hash`	Target entry's Event ID: unpadded base64url of its 32-byte line hash
`attester_key`	Unpadded base64url of the 33-byte SEC1 compressed credential public key
`assertion_sig`	Unpadded base64url of the 64-byte WebAuthn signature, raw `r ‖ s`
`authenticator_data`	Exact authenticator-data bytes, unpadded base64url
`client_data_json`	Exact client-data JSON bytes, unpadded base64url

A Mint entry declares `v = "1"` and carries all six Mint members beside the base members. Partial or misplaced Mint members, or an unknown `kind`, make the entry malformed. The target must be an earlier verified entry in the same chain.

WebAuthn Assertion Checks

For a Mint entry, the verifier checks three things, following the WebAuthn assertion format (Hodges et al. 2021). `client_data_json` must be valid UTF-8 JSON of type `webauthn.get` whose challenge equals the encoded `target_hash`. `authenticator_data` must hold at least 37 bytes, with the user-presence flag set and consistent backup-state flags. `assertion_sig` must verify under `attester_key` over `authenticator_data ‖ SHA-256(client_data_json)`. These checks leave the relying party's origin and RP ID policy, user verification and signature counter to the application's accepted authentication context.

Heartbeat Encoding and Retrieval

Signed format. The Hub takes `position` and `head_hash` in Table 6.3 from the same chain state.

Table 6.3: Heartbeat fields

Member	Meaning
`v`, `kind`	Literal values `"1"` and `"heartbeat"`
`topic_id`	Exact topic identifier from the agreed settings
`position`	Number of entries in the committed prefix, at least one
`head_hash`	SHA-256 of the exact final row in that prefix, excluding its terminating `0x0a`
`issued_at`	Hub-asserted issuance time, Unix seconds with an optional fractional part
`signature`	Hub signature over the statement fields above

The encoding is a UTF-8 JSON object with string values. The Hub signs the JCS serialisation of the other members and then adds `signature`, using ECDSA-P256 and the binary encodings of the entry specification. A verifier rejects duplicate or unknown members, unknown revisions, unpaired surrogates and malformed values. `position` uses ASCII decimal digits without leading zeros. `issued_at` uses the same form, optionally followed by a decimal point and fractional digits without trailing zeros. The verifier takes the accepted Hub key and interval from the agreed publication settings.

Retrieval. The Hub serves these endpoints for hosts to pull, and each host serves them for holders. Relative to the server's base URL for the topic, `GET heartbeat/latest` returns the latest heartbeat as one JSON object and `GET heartbeat` returns the full list as an array. `GET heartbeat?from={timestamp}&until={timestamp}` returns the heartbeats with `from <= issued_at < until`, and an omitted bound leaves that end open. If conflicting signed heartbeats arise, the host keeps both.

Submission, Receipt and Refusal

Submission. A submission is one or more lines, each the exact bytes of an entry, joined by `0x0a` with no terminator after the last. The first line's `prev_hash` is the expected head, and no other field carries it; each later line's `prev_hash` names the hash of the

line before it in the submission. A line is a JSON object and cannot contain a raw newline, so the separator is unambiguous; a body that ends in `0x0a` has an empty last line and counts as malformed. The Hub appends the body and one `0x0a`, whole or not at all. The *submission hash* is the SHA-256 digest of the body, so for a one-line submission it is the line's Event ID.

Receipt and refusal. A receipt or refusal has the shape of a heartbeat, so a verifier that checks one checks all three: a flat JSON object of string members, signed under the Hub key over the JCS serialisation of the other members, with the encodings of the entry specification and the rejection rules of the heartbeat. Table 6.4 lists the members.

Table 6.4: Receipt and refusal fields

Member	Receipt	Refusal
`v`, `kind`	`"1"`, `"receipt"`	`"1"`, `"refusal"`
`topic_id`	Exact topic identifier	Exact topic identifier
`submission_hash`	Hash of the submitted body	Hash of the submitted body
`issued_at`	Hub-asserted time, as in a heartbeat	Hub-asserted time, as in a heartbeat
`position`, `head_hash`	Chain state after the append; `head_hash` is the last appended line's Event ID and binds every line before it	Present on `head_mismatch`: the current state, so the submitter knows what extension to fetch
`count`	Number of lines appended	Absent
`reason`	Absent	One value from Table 6.5
`line`	Absent	On a verifier fault, the ordinal of the refused line within the body
`signature`	Hub signature over the members above	Hub signature over the members above

Table 6.5 lists the reasons a refusal can carry. The first five are the verifier's own fault names, so a refusal and a verifier verdict on the same line agree; the last is the Hub's.

Table 6.5: Refusal reasons

Reason	Meaning
`malformed`	The line is not an entry under the entry specification
`chain_mismatch`	The line's `prev_hash` is not the hash of the line before it in the submission

Reason	Meaning
`key_off_curve`	The `public_key` is 33 well-formed bytes that name no point on P-256
`signature_invalid`	The signature is not this key's over the line's signed message
`attestation_fault`	A Mint envelope failed one of the WebAuthn checks
`head_mismatch`	The first line's `prev_hash` is not the current head; another submission arrived first

A receipt is the Hub's signed commitment that the submission is in the history at that position, so a history that later omits it contradicts the Hub's own signature, and the producer who retained the receipt holds the proof. The Hub never retries a refused submission and never reuses a signature over an old head: the submitter fetches the extension, verifies it and signs again.

CBC and Correction References

Offer and acceptance. The offer is an ordinary entry whose payload names the subject of the agreement, the parties, the conditions and the time of the offer. The acceptance carries `subject_hash`, the offer's Event ID, together with its acknowledgement of the terms. The reference binds the acceptance to that exact offer wherever the acceptance sits in the chain. When the parties require the acceptance to follow the offer immediately, its `prev_hash` also equals the offer's Event ID. Whether `subject_hash` becomes an entry member or a payload field remains open, and it belongs to neither the base nor the Mint members above. Either way, the signature must cover it, directly or through the payload hash.

Corrections

A statement that addresses an earlier entry refers to it through `subject_hash`, exactly as an acceptance does, while `prev_hash` still names the immediately preceding line. The payload says which kind of statement the entry is and carries its substance. When the accepting party disputes an offer, it withholds acceptance and records a signed objection that refers to the offer, so the CBC stays pending and the objection says why; a resolution is a further signed entry that refers to the objection. When an author has

lost its key and enrolled a new one (Section 4.3), application policy decides whether a correction signed with the new key counts as that author's.

Key Registries

A registry (Section 4.3) is a topic whose entries are binding events and whose *keeper* accepts each binding. Each is an ordinary entry under the entry specification with no additional member. Four kinds of binding event exist, and the payload says which kind an entry is:

- Register. The subject key signs the offer, so the entry's `public_key` is the key the binding covers. A person's passkey cannot sign an entry, so for a person the producing system signs the offer and the person confirms it in a Mint entry (WebAuthn checks), and the binding covers `attester_key`. The payload carries the subject type (person, device, service or registry root), an identity reference, a role and the hash of the enrolment evidence. The keeper signs the acceptance with `subject_hash` naming the offer (CBC references). The binding stands from the acceptance's position; before that it is pending, and a refusal is a signed objection (corrections).
- Rotate. The new key signs the offer, whose payload names the binding it replaces by Event ID, and the keeper accepts. A registry's root key rotates the same way: the new root signs the offer and the current root accepts, and holders follow the new root from that position.
- Retire. The keeper signs one entry that names the binding by Event ID. Signatures the key made before that position stand.
- Compromise. The keeper signs one entry that names the binding and the time from which the key is suspect, which may precede the entry. The verifier reports each entry the key signed at or after that time and before the claim's position as inside a claimed compromise window and leaves the decision to the application.

Under the payload rule, the binding payload travels with its salt, so a holder resolves keys offline; the enrolment evidence, including any personal details, stays with the keeper under its access policy. Each topic declares the registries it accepts, by registry topic and root key, directly or through a binding in a registry it already accepts, and a change to that declaration is a CBC that the topic's other parties countersign. When two accepted registries bind one key differently, the verifier reports conflicting, keeps both bindings and leaves the decision to the application.

Three conventions remain open, each with the constraint an answer must meet:

- Declaration encoding. How the first entry carries the accepted registries and the scope of each, and how a holder authenticates which topic a history belongs to

through that first entry. Constraint: every holder can read it without the payload access policy.
- Entry-to-binding reference. Whether a verifier resolves a key by lookup in the accepted registries at verification time, as this appendix assumes, or an entry carries a reference to its binding. Constraint: a carried reference needs a new entry revision, because an unknown member makes an entry malformed.
- Attribution outcomes. The states the verifier reports for a key, modelled on the heartbeat outcomes: bound, pending, unbound, conflicting, retired before signing and inside a claimed compromise window are the candidates. Constraint: signature validity, binding evidence, compromise status and authorisation stay separate.

The foundation stewards these conventions (Section 5.1). The questions it settles first:

1. Who may keep a registry that others accept, and what enrolment evidence each must demand.
2. How a keeper recovers when it loses its root key or a thief rotates the root first: a new first entry, and each accepting topic's countersigned change to its declaration.
3. Whether any rule beyond reporting decides between two accepted registries that disagree.
4. How far back a compromise claim may reach, and who must countersign one.
5. How long a keeper retains enrolment evidence, and how it deletes it.

Payload Access and Export

The application attaches the access policy to the payload, never to `payload_ref`. An authorised holder can export public payloads with their salts, keeping the exact bytes the hash covers and linking each to its `payload_ref` or Event ID, so others can verify them offline. Salt length and encoding, including the boundary between salt and payload, remain open in the specification. A public key or a payload hash can still be linkable to a person, so neither is automatically anonymous under GDPR.

Verifier Outcomes for Heartbeat Evidence

The verifier reports heartbeat evidence as missing when no authenticated heartbeat for the topic is retrievable, stale when the latest one falls outside the application's freshness window, or conflicting when authenticated heartbeats or the supplied history disagree at a covered position. For that topic, the application pauses the actions that need verified shared evidence, keeps capturing and queuing new events locally, and escalates. A conflict keeps both heartbeats for reconciliation.

Appendix: Comparison Tables

This appendix holds the comparison tables that Chapter 3 relies on and shows how EventChain relates to existing transparency and provenance systems.

Standards Lineage

EventChain sits in a lineage of cryptographic accountability systems. Haber and Stornetta established hash-linked time-stamping as a way to make document history tamper-evident without a conventional record-keeper (Haber and Stornetta 1991). Certificate Transparency and Trillian turned append-only transparency logs into operational infrastructure for public certificates (Laurie et al. 2013; Google 2026). SCITT generalises transparency receipts for supply-chain statements using COSE/CBOR and external transparency services (Birkholz et al. 2026). KERI focuses on self-certifying identifiers and key-event logs for decentralised identity (Smith 2019). Parakeet demonstrates privacy-preserving transparency through public commitments, private querying, and verifiable proof (Malvai et al. 2023). TrustChain shows that a blockchain can preserve data while leaving truth at origin unresolved (Malik et al. 2019).

EventChain keeps the chained hash and the published heartbeat from these systems. What it adds is a key for every signer and a file the parties keep themselves.

Table 6.6: Provenance and transparency architectures compared

Property	EventChain	SCITT
Primary object	Signed event history	Supply-chain statement receipt
Attribution	Signing key plus enrolment evidence	Entity via X.509 or OIDC
Proof location	Travels with authorised holders	Receipt can travel with signed statement
Verification	Offline chain/signatures; separate binding and heartbeat evidence	Receipt, accepted service keys and proofs

Property	EventChain	SCITT
Tooling	JSON Lines, SHA-256, ECDSA-P256, JCS and passkeys	COSE/CBOR, verifiable structures and registry
Storage/distribution	Hub distribution; application payload access	Left to implementers
Access control	Application policy over encrypted payloads	Out of scope
Two-party agreement	Offer and signed acceptance (Section 4.6)	Application-defined

Table 6.7: Earlier transparency and identity systems, using the same comparison properties

Property	CT / Trillian	Haber & Stornetta	KERI
Primary object	Certificate log entry	Time-stamped document hash	Key-event history
Attribution	Certificate authority / subject	Requester identifier in the time-stamp certificate	Identifier controller
Proof location	External log	External time-stamp chain	Key-event log that travels with the identifier
Verification	Inclusion / consistency proof	Chain through time-stamp service	Key-event log verification
Tooling	Merkle log infrastructure	Hash chain and time-stamp service	KEL and witness infrastructure
Storage/distribution	Outside certificate-log scope	Outside protocol scope	Outside protocol scope
Access control	Public log model	Out of scope	Out of scope
Two-party agreement	Outside scope	Outside scope	Outside scope

SCITT and EventChain operate at different layers. SCITT answers whether a transparency service registered a signed statement; its receipt can accompany the statement for verification (Birkholz et al. 2026, sec. 7). EventChain preserves producer-signed entries and their sequence in a portable file.

Dishonest Hub

The Hub accepts producer-signed entries, orders them and serves authorised copies. Table 6.8 sets out what a dishonest Hub can attempt, how a holder detects it and what risk remains. Figure 4.2 shows the comparison each holder makes.

Table 6.8: Adversarial Hub failure modes, detection paths, and residual risks

Failure Mode	Detection Path	Residual Risk
Hub alters an old entry	Chain and producer-signature checks expose alterations; a retained head or a later heartbeat also binds exact final-entry bytes	A party must run verification; hash checks alone authenticate nothing without the producer signatures
Hub withholds an entry from a role-authorised party	A supplied copy falls short of or conflicts with the published heartbeat	The file cannot establish receipt of an unacknowledged submission; missing or stale heartbeats alone do not prove withholding
Hub binds a valid signature to wrong payload	Holder recomputes the salted payload hash from the plaintext and salt	Parties lacking the exact plaintext and salt cannot test that hash
Hub presents different histories to different holders (a fork)	Each holder compares its copy with a heartbeat host the foundation lists for its topic; conflicting Hub-signed heartbeats attribute the fork to the Hub's key	Detection requires a host that keeps every heartbeat and shows everyone the same list; fresh entries await the next heartbeat

If no host listed for the topic (Section 4.4.1) is reachable, the file alone provides fork consistency: incompatible branches become detectable when holders compare copies (Li et al. 2004). OpenTimestamps receipts cannot stand in for the list (Section 4.8).

The Hub's one remaining duty, staying available to accept new entries and serve copies, rests on the Hub service provider's service agreement.

Trust Boundary

Holders verify integrity and order from the file. Naming signers and checking freshness need external evidence, and applications govern payload access and decisions (Table 6.9).

Table 6.9: Trust boundary in the EventChain architecture

Capability group	Evidence or responsibility	Remaining dependency
Holder checks: integrity and order	Verify links and producer signatures offline	Correct verifier and retained AOF
Holder checks: payload commitment	Hash raw salt followed by exact plaintext bytes	Authorised access to plaintext and salt
External evidence: authorship	Verify key signature; assess identity binding (Table 4.2)	Retained enrolment and dispute evidence
External evidence: freshness/shared history	Compare with authenticated heartbeat (Figure 4.2)	Accepted Hub key; preserved same list; publication interval
Operations: appending/distribution	Hub accepts entries and serves copies	Availability; application-controlled payload access
Operations: search/indexing	Hub supplies discovery and query results	Check returned evidence against retained history
Operations: portal views	Application presents selected evidence	Presentation and action policy remain application duties

Failure Modes

The following tables separate what the evidence establishes from what remains unknown and requires operating controls.

Table 6.10: History integrity and verification

Failure mode	What the evidence establishes	What remains unknown	Complementary control
Tampered entry (post-signing edit)	Chain or producer-signature checks fail (Table 6.8)	Checks do not necessarily locate the original alteration	Routine verification by holders
Verification never run	None	A valid file that no party bothers to check	Contractual verification obligations, automated verification triggers

Table 6.11: Signer conduct

Failure mode	What the evidence establishes	What remains unknown	Complementary control
False data from an authorised signer	Signature identifies the key; enrolment evidence supports attribution	The falsehood itself, and the intent behind it	Process controls, cross-checks, independent measurement, separation of duties
Honest signer error	None	Mistakes that produce valid-looking entries	Quality systems, peer review, calibration
Coerced signature	None	Duress behind a technically valid signature	Whistleblower channels, dual-control policies
Rubber-stamping	None	Perfunctory signatures without genuine review	Audit culture, random re-inspection

Table 6.12: Credentials and access

Failure mode	What the evidence establishes	What remains unknown	Complementary control
Compromised passkey	None; a later signed compromise claim stands beside the history as a visible dispute	A signature produced from a stolen or cloned credential	Hardware-bound passkeys with user verification, device attestation, anomaly detection
Lost or unrecoverable key	Earlier signatures stay valid; the organisation retires the key and issues a new one	None	Key-loss procedures, identity re-proofing
Restricted payload access	Chain and signature checks still verify the history	Content of payloads the holder cannot access	Application access policy and retention of authorised supporting evidence

Table 6.13: Physical measurement

Failure mode	What the evidence establishes	What remains unknown	Complementary control
Sensor removed from the asset it monitors	A CBC signed at installation by the technician and the sensor (Section 4.6) records which sensor the technician fitted to which asset	A removed sensor keeps signing valid readings, because the key is in the sensor and shows nothing about where the sensor is	Physical tamper indicators and inspection
Sensor calibration drift	None	Gradual measurement error within technically valid signatures	Calibration schedules, reference comparisons

Choosing an Approach

Table 6.14 sets EventChain beside the other ways of keeping provenance. The approaches combine: an organisation might keep ERP audit trails internally and use EventChain for the evidence shared across organisations.

Table 6.14: Approaches to provenance: when each fits and where it falls short

Approach	Works well when	Falls short when	Adoption condition
Do nothing	Single-party control, low stakes, short lifecycles, negligible dispute risk	Multiple handoffs, regulatory scrutiny, or long-lived histories where reconstruction costs compound	Risk exposure remains below the cost of any provenance system
ERP audit trails	All relevant parties share one system, or a dominant buyer mandates platform access	Records must survive beyond the platform operator, or partners lack ERP access	Organisational control spans the full lifecycle
Document signing (e-signatures)	Point-in-time attestation between known parties, such as contract execution or inspection sign-off	Continuity matters, because signed documents do not chain into a verifiable sequence across several actors	The proof need is a single moment
Managed compliance portals	Compliance requires a portal with public access and standardised data models	The portal cannot produce independently verifiable evidence, or the history must outlive the platform vendor	A single vendor can credibly serve the history's full lifetime
VC/DID credentials	Interoperable identity claims across organisations, where selective disclosure and the subject's control of its credentials matter	The question is the order and integrity of events, which credentials do not carry	Identity portability is the primary requirement

Approach	Works well when	Falls short when	Adoption condition
Enterprise distributed ledger	High-value, multi-party transactions that need shared state no single operator controls	Consortium formation stalls, costs exclude smaller participants, or throughput requirements exceed consensus limits	All parties can fund node infrastructure and governance
SCITT transparency registry	Public accountability for signed statements, with receipts that can travel with those statements	Statement storage, discovery, and distribution need an implementation beyond the registry specification	A transparency service exists and holders accept its verification keys and policies
EventChain with an application layer	Identified parties in different organisations need a long-lived history each can verify, including participants who cannot fund ledger infrastructure	Strangers settle payments with one another, or one organisation controls the whole process and nobody outside needs to verify	At least one party needs proof that survives beyond any single platform

Deployment Dependencies

The following tables list what a production deployment puts in place beyond the file and the verifier (Section 5.2).

Table 6.15: Evidence production and availability

Dependency	Owner	Why it matters	Risk if ignored
Hub (hosted or self-run)	Hub service provider	Appends entries, distributes the AOF, enforces payload access policy	No chain creation or distribution
Heartbeat publication	Hub service provider	Signs and publishes heartbeats at the topic's agreed interval	Holders cannot tell whether their copy is current

Dependency	Owner	Why it matters	Risk if ignored
Heartbeat hosting and accepted Hub keys	Heartbeat hosts the foundation lists for the topic	Pull each heartbeat from the Hub, check it, preserve it; serve the same list and accepted Hub keys to everyone	Holders cannot establish freshness or reliably detect a fork

Table 6.16: Identity and access

Dependency	Owner	Why it matters	Risk if ignored
Identity provider integration	IT / Security	Maps passkeys to verified personnel	Unsigned or unattributable events
Signer onboarding	HR / IT	Registers passkeys against identity	Adoption stalls at first signing step
Payload access policy	Governance / Legal	Controls authorised payload access	Over-sharing or under-sharing halts rollout

Table 6.17: Application decisions and use

Dependency	Owner	Why it matters	Risk if ignored
Business logic and event schema	Domain / Engineering	Defines which actions become evidence and what each entry records	Unverifiable payloads
Holder verifier access	Holder IT	Enables independent verification (Table 4.3)	Proof exists but nobody checks it
ERP/API webhook integration	Engineering	Connects evidence to existing workflows	Manual data entry, adoption friction
Contractual verification clause	Legal / Procurement	Defines consequences of failed verification	A failed verification carries no consequence

References

Abadi, Daniel J. 2012. "Consistency Tradeoffs in Modern Distributed Database System Design: CAP Is Only Part of the Story." *Computer* 45 (2): 37–42. https://doi.org/10.1109/MC.2012.33.

Accenture. 2022. *ASX CHESS Replacement Application Delivery Review*. Accenture, published by the Australian Securities; Investments Commission (Report 23-035). https://download.asic.gov.au/media/ku2aa1kz/23-035-asx-chess-replacement-application-delivery-review-2022.pdf.

Androulaki, Elli, Artem Barger, Vita Bortnikov, et al. 2018. "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains." *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*. https://doi.org/10.1145/3190508.3190538.

A.P. Moller - Maersk. 2022. *A.p. Moller - Maersk and IBM to Discontinue TradeLens, a Blockchain-Enabled Global Trade Platform*. https://www.maersk.com/news/articles/2022/11/29/maersk-and-ibm-to-discontinue-tradelens.

ascribe team. n.d. *Ascribe Is No Longer Active*. Accessed October 12, 2026. https://www.ascribe.io/.

ASX Limited. 2022. *CHESS Replacement: ASX Reassessing Solution Design and Financial Derecognition*. Market release, reproduced by Mondo Visione. https://mondovisione.com/media-and-resources/news/chess-replacement-asx-reassessing-financial-derecognition-20221117/.

Birkholz, Henk, Antoine Delignat-Lavaud, Cédric Fournet, Yogesh Deshpande, and Steve Lasker. 2026. *An Architecture for Trustworthy and Transparent Digital Supply Chains*. RFC 9943. https://doi.org/10.17487/RFC9943.

Cisco. 2024. *More Than 1 in 4 Organizations Banned Use of GenAI over Privacy and Data Security Risks*. https://newsroom.cisco.com/c/r/newsroom/en/us/a/y2024/m01/organizations-ban-use-of-generative-ai-over-data-privacy-security-cisco-study.html.

CNN. 2024. *British Engineering Giant Arup Revealed as $25 Million Deepfake Scam*

Victim. https://edition.cnn.com/2024/05/16/tech/arup-deepfake-scam-loss-hong-kong-intl-hnk.

Coalition Against Insurance Fraud. 2022. *The Impact of Insurance Fraud on the U.S. Economy*. https://insurancefraud.org/wp-content/uploads/The-Impact-of-Insurance-Fraud-on-the-U.S.-Economy-Report-2022-8.26.2022.pdf.

Cooper, Brian F., Raghu Ramakrishnan, Utkarsh Srivastava, et al. 2008. "PNUTS: Yahoo!'s Hosted Data Serving Platform." *Proceedings of the VLDB Endowment* 1 (2): 1277–88. https://doi.org/10.14778/1454159.1454167.

Dubai Land Department. 2025. *Dubai Land Department Launches Pilot Phase of the Real Estate Tokenisation Project*. https://dubailand.gov.ae/en/news-media/dubai-land-department-launches-pilot-phase-of-the-real-estate-tokenisation-project.

Energinet. 2023. *Project Origin: Connecting the Dots from Energy Production to Consumption*. LF Energy Summit presentation. https://static.sched.com/hosted_files/lfenergysummit2023/e0/Project%20Origin%20pitch_deck_Polygraphic_Ver3_Paris_Nicolas_Bernhardi_Energinet.pdf.

Etherscan. 2026. *Ethereum Daily Transactions Chart*. https://etherscan.io/chart/tx.

European Parliament and Council. 2024. *Regulation (EU) 2024/1781 of 13 June 2024 Establishing a Framework for the Setting of Ecodesign Requirements for Sustainable Products, Amending Directive (EU) 2020/1828 and Regulation (EU) 2023/1542 and Repealing Directive 2009/125/EC*. Official Journal of the European Union, L, 28.6.2024. https://eur-lex.europa.eu/eli/reg/2024/1781/oj.

EventChain. 2026a. *EventChain: Open Haskell Implementation*. Https://github.com/eventchain-protocol/eventchain.

EventChain. 2026b. *Hub Append Rate: What Checking Each Line Before Appending It Costs*. Reproducible reference-implementation benchmark.

EventChain. 2026c. *Minimal Actual-File Verifier Evidence*. Reproducible reference-implementation benchmark.

EventChain. 2026d. *Producer Batching: What Signing Events in Batches Buys Under Contention*. Reproducible reference-implementation benchmark.

EventChain Protocol. 2026. *EventChain Hub Reference Implementation*. GitHub repository. https://github.com/eventchain-protocol/hub.

FIDO Alliance. 2026. *FIDO Alliance Reports Accelerating Global Passkey Adoption on World Passkey Day 2026*. https://fidoalliance.org/fido-alliance-reports-accelerating-global-passkey-adoption-on-world-passkey-day-2026/.

Google. 2026. *Trillian: General Transparency*. GitHub repository. https://github.com/google/trillian.

Gottlieb, Glenn. 2026. *The Quiet Winners: Enterprise Blockchain Applications That Deliver Durable Value*. Ecosystem Insights. Architect Partners. https://architectpartners.com/wp-content/uploads/2026/02/The-Quiet-Winners_Enterprise-Blockchain-Application-That-Deliver-Durable-Value.pdf.

Haber, Stuart, and W. Scott Stornetta. 1991. "How to Time-Stamp a Digital Document." *Journal of Cryptology* 3 (2): 99–111. https://doi.org/10.1007/BF00196791.

Hill, Rebecca. 2018. *IBM Struggles to Sign up Shipping Carriers to Blockchain Supply Chain Platform*. The Register. https://www.theregister.com/2018/10/30/ibm_struggles_to_sign_up_shipping_carriers_to_blockchain_supply_chain_platform_reports/.

HM Land Registry. 2026. *Practice Guide 67: Evidence of Identity*. https://www.gov.uk/government/publications/evidence-of-identity-conveyancers/practice-guide-67-evidence-of-identity-conveyancers.

Hodges, Jeff, J. C. Jones, Michael B. Jones, Akshay Kumar, and Emil Lundberg. 2021. *Web Authentication: An API for Accessing Public Key Credentials, Level 2*. W3C Recommendation. https://www.w3.org/TR/webauthn-2/.

Hong Kong Free Press. 2024. *Multinational Loses HK$200 Million to Deepfake Video Conference Scam, Hong Kong Police Say*. https://hongkongfp.com/2024/02/05/multinational-loses-hk200-million-to-deepfake-video-conference-scam-hong-kong-police-say/.

Howard, L. S. 2022. *Industry's Blockchain Project, B3i, Ceases to Trade After Filing for Insolvency*. Carrier Management. https://www.carriermanagement.com/news/2022/08/01/238818.htm.

IBM. 2021. *Hapag-Lloyd and Ocean Network Express Complete TradeLens Integration*. IBM Newsroom. https://newsroom.ibm.com/2021-06-24-Hapag-Lloyd-and-Ocean-Network-Express-Complete-TradeLens-Integration,-Join-Rapidly-Expanding-Shipping-Ecosystem-to-Further-Enhance-Trade-Digitization.

Josefsson, Simon. 2006. *The Base16, Base32, and Base64 Data Encodings*. RFC 4648.

https://doi.org/10.17487/RFC4648.

JSON Lines. 2023. *JSON Lines*. Https://jsonlines.org/.

Laurie, Ben, Adam Langley, and Emilia Kasper. 2013. *Certificate Transparency*. RFC 6962. https://doi.org/10.17487/RFC6962.

Li, Jinyuan, Maxwell Krohn, David Mazières, and Dennis Shasha. 2004. "Secure Untrusted Data Repository (SUNDR)." *6th Symposium on Operating Systems Design and Implementation (OSDI)*, 121–36. https://www.usenix.org/legacy/event/osdi04/tech/full_papers/li_j/li_j.pdf.

Malik, Sidra, Volkan Dedeoglu, Salil S. Kanhere, and Raja Jurdak. 2019. "TrustChain: Trust Management in Blockchain and IoT Supported Supply Chains." *IEEE International Conference on Blockchain*. https://doi.org/10.1109/Blockchain.2019.00032.

Malvai, Harjasleen, Lefteris Kokoris-Kogias, Alberto Sonnino, et al. 2023. "Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging." *Network and Distributed System Security Symposium (NDSS)*. https://www.ndss-symposium.org/ndss-paper/parakeet-practical-key-transparency-for-end-to-end-encrypted-messaging/.

McGee, Scott. 2022. *B3i Enters Insolvency After Failing to Raise Funding*. Insurance Post. https://postonline.co.uk/technology/7950336/b3i-enters-insolvency-after-failing-to-raise-funding.

Microsoft. 2026. *Windows 11 Requirements*. https://learn.microsoft.com/en-us/windows/whats-new/windows-11-requirements.

Najati, Issam. 2025. "Exploring the Failure Factors of Blockchain Adopting Projects: A Case Study of TradeLens Through the Lens of Commons Theory." *Frontiers in Blockchain* 8. https://doi.org/10.3389/fbloc.2025.1503595.

Nasr, Milad, Nicholas Carlini, Jonathan Hayase, et al. 2023. *Scalable Extraction of Training Data from (Production) Language Models*. arXiv:2311.17035. https://arxiv.org/abs/2311.17035.

National Institute of Standards and Technology. 2015. *Secure Hash Standard (SHS)*. FIPS Nos. 180-4. NIST. https://doi.org/10.6028/NIST.FIPS.180-4.

National Institute of Standards and Technology. 2023. *Digital Signature Standard (DSS)*. FIPS Nos. 186-5. NIST. https://doi.org/10.6028/NIST.FIPS.186-5.

OpenTimestamps. 2026. *OpenTimestamps: A Timestamping Proof Standard*. https://opentimestamps.org/.

PierNext. 2023. *The Closure of TradeLens: When Technology Is Not Enough*. Port de Barcelona. https://piernext.portdebarcelona.cat/en/technology/the-closure-of-tradelens/.

Reddy, Raksha, and Carl Wallace. 2010. *Trust Anchor Management Requirements*. RFC 6024. https://doi.org/10.17487/RFC6024.

Rundgren, Anders, Bret Jordan, and Samuel Erdtman. 2020. *JSON Canonicalization Scheme (JCS)*. RFC 8785. https://doi.org/10.17487/RFC8785.

Serious Fraud Office. 2026. *SFO Secures 4-Year Prison Sentence for Aircraft Parts Fraud*. https://www.gov.uk/government/news/sfo-secures-4-year-prison-sentence-for-aircraft-parts-fraud.

Smith, Samuel M. 2019. *Key Event Receipt Infrastructure (KERI)*. https://arxiv.org/abs/1907.02143.

Standards for Efficient Cryptography Group. 2009. *SEC 1: Elliptic Curve Cryptography, Version 2.0*. Certicom Research. https://www.secg.org/sec1-v2.pdf.

Tabak, Nate. 2020. *How Walmart Solved Carrier Payment Woes in Canada with Blockchain*. FreightWaves. https://www.freightwaves.com/news/how-walmart-solved-carrier-payment-woes-in-canada-with-blockchain.

Todd, Peter. 2016. *OpenTimestamps: Scalable, Trust-Minimized, Distributed Timestamping with Bitcoin*. https://petertodd.org/2016/opentimestamps-announcement.

TradeLens. 2021. *Pricing*. https://www.tradelens.com/pricing.

UK Parliament. 2023. *Electronic Trade Documents Act 2023*. https://www.legislation.gov.uk/ukpga/2023/38/enacted.

United Nations Commission on International Trade Law. 2025. *UNCITRAL Model Law on Electronic Transferable Records (2017): Status*. https://uncitral.un.org/en/texts/ecommerce/modellaw/electronic_transferable_records/status.

United Nations ESCAP. 2025. *TradeLens*. Cross-Border Paperless Trade Database (Digitalize Trade). https://www.digitalizetrade.org/projects/tradelens.

U.S. Customs and Border Protection. 2026. *Uyghur Forced Labor Prevention Act*. https:

//www.cbp.gov/trade/forced-labor/UFLPA.

U.S. Food and Drug Administration. 2018. *Data Integrity and Compliance with Drug CGMP: Questions and Answers, Guidance for Industry*. U.S. Food; Drug Administration. https://www.fda.gov/media/119267/download.

Wragg, Eleanor. 2022a. *Trade's Blockchain Proponents Face Reality Check as Hype Fades*. Global Trade Review. https://www.gtreview.com/magazine/the-fintech-issue-2022/trades-blockchain-proponents-face-reality-check-as-hype-fades.

Wragg, Eleanor. 2022b. *We.trade Calls It Quits After Running Out of Cash*. Global Trade Review. https://www.gtreview.com/news/top-stories/we-trade-calls-it-quits-after-running-out-of-cash/.

Wragg, Eleanor. 2023a. *Marco Polo Brings in Liquidators as Funds Run Dry*. Global Trade Review. https://www.gtreview.com/news/fintech/marco-polo-brings-in-liquidators-as-funds-run-dry/.

Wragg, Eleanor. 2023b. *Shipping Industry Sets Course for Fully Paperless Trade*. Global Trade Review. https://www.gtreview.com/news/fintech/shipping-industry-sets-course-for-fully-paperless-trade/.