

09

Nobody gets to redo the paperwork

Every party in a shipment's history signs what it saw, and any single fact can be proved without opening the rest.

BUILD STATE

Yours to take

COMPLEXITY



Vision

A jar on a shelf says organic, single-origin, Colombia. Behind that sentence is a certificate, issued by a body the producer selected and paid, renewed on an annual inspection, and held as a PDF by the company whose product it vouches for. The rest of the history is somewhere else entirely: a farm's own notes, a cooperative's weighbridge tickets, a shipping line's manifests, a processing plant's batch records, each in a different system belonging to a different company with a different reason to present itself well. Nobody holds the whole picture. Everyone holds their own piece of it, and their piece is the only version anyone outside will ever see.

That arrangement works until something goes wrong. When products labelled as beef across Europe turned out to contain horsemeat in 2013, the difficulty was not discovering the fraud, which a routine test found. The difficulty was tracing it. The chains ran through traders and intermediaries in several countries, each holding its own paperwork, and reconstructing what had actually happened took months of subpoenas and cooperation. The records existed. They were just scattered across the very companies with the strongest interest in how they read.

The structural problem is not that supply chain records are missing. It is that each one is held by the party it describes. A company that later finds a record inconvenient can revise it, lose it in a migration, or simply decline to produce it, and there is rarely a way for anyone outside to tell which of those happened. Auditors work around this by sampling, which catches routine sloppiness and misses anything deliberate. Certification works around it by inspecting once a year and trusting the other three hundred and sixty-four days.

There is a second problem underneath, and it is the one that keeps honest parties quiet. Proving a single claim usually means handing over everything around it. A buyer asks whether a batch ever went above four degrees in transit; answering with the shipment log also hands over the routing, the volumes, the other customers on the same truck. So the producer offers a summary and asks to be believed, which is where the whole exercise started. Disclosure is all or nothing, so most people choose nothing.

Swarm changes both halves. Each party writes its own record into storage that no company operates, signed with its own key and addressed by its contents, so no party can write into another's record and no party can revise its own past entries after learning what they turned out to prove. And because those records are written in a format designed to be checked in pieces, a single fact out of one can be shown to belong to it, on its own, with the rest of the document staying shut. The temperature answer can be given without the routing attached.

What this buys is narrower than the phrase "verified provenance" usually implies, and the difference matters. None of it makes a record true. A farm can sign a false reading; a plant can sign a yield it did not achieve. What changes is that the claim has to be committed to at the time, in public, under a named key, before anyone knows which parts will later matter, and it then sits permanently beside everyone else's account of the same events. A producer who has signed the same readings honestly for six years has built something an audit cannot hand out and a competitor cannot fabricate.

Architecture

The unit of this space is a record: a small typed object, written by one party, whose bytes have exactly one valid form, whose address is derived from those bytes, and which can be interrogated a field at a time without being opened. Everything below serves that.

Canonical bytes (BeeSon; see the maturity note closing this section). A provenance record is a hash commitment, so its serialization has to be canonical, and JSON is not. The same object can be written many ways, key order and whitespace and number formatting all free, so hashing JSON directly produces a value that depends on who wrote it out. BeeSon is a JSON-compatible binary format that fixes this: key order follows the schema's own type-definition order and padding is deterministic, so the same data under the same schema always produces the same bytes and therefore the same address. Two parties encoding the same reading independently arrive at the same hash.

Types that survive the trip (BeeSon). JSON has one number type. BeeSon has signed and unsigned integers at four widths, two float widths, strings, booleans, nulls, and arrays and objects in strict and nullable variants, and it enforces them on assignment: writing a decimal into a field typed as an integer throws rather than silently rounding somewhere downstream. For sensor data, where a temperature, a mass and a pH are different kinds of quantity, the type is part of what the record means. Two of the types are Swarm addresses, a manifest reference and a feed reference, so a record that points at another Swarm object holds a typed reference rather than a string that happens to look like a hash.

Proving one field instead of publishing the document (BeeSon + BMT). BeeSon pads everything to 32-byte segments, the schema section padded as a whole so that the data begins on a fresh segment and each value occupies whole segments of its own. The layout is chosen for exactly one reason, and the format's own documentation gives it: a value sitting at a known segment offset can be proved to belong to a chunk by a Binary Merkle Tree inclusion proof against that chunk's address, and the proof is small. So the holder of a record can demonstrate that one field of it has a particular value, checkable against the address the record was signed under, while the rest of the record stays undisclosed. That is selective disclosure achieved by proof rather than by encryption, with no keys to distribute and nothing to revoke, and it turns an all-or-nothing disclosure into a specific answer to a specific question.

A schema with an address of its own (BeeSon). A container's type specification can be stored once and referenced by a 32-byte Swarm hash instead of repeated inside every record, so a record reduces to a 32-byte header, a 32-byte reference to its schema, and its data. Ten thousand batch records on a shared schema stop repeating field names, and the inclusion proof gets cheaper because there is less in front of the data. The more useful consequence is governance: the schema becomes an addressable object, so which version of a record format a given record was written against is answerable by hash rather than by asking. In a supply chain where several organisations have to agree on what a field means, that is the object they are actually agreeing on.

One feed per signing party (shipped). Records are anchored in Swarm feeds, each a sequence of Single Owner Chunks signed by an Ethereum keypair and identified by the owner's address together with a 32-byte topic derived from a name. Derive the topic from the batch identifier and each party's entries for that batch live at an address that follows from public information, so a reader who knows a party's address and the batch identity computes where to look without a registry or an index service. Events append at the next index, earlier entries stay put, and every entry's signature establishes which key wrote it. The farm cannot append to the carrier's feed; the carrier cannot rewrite the farm's. Cross-referencing happens in the reader, not in a system anyone operates.

The transformation event. Most provenance systems track a pallet that never changes, which is the easy case. The hard case is where inputs stop existing and an output begins: a freeze-drying plant

takes several hundred kilos of fresh blueberries and produces a much smaller mass of dried product, with moisture loss, waste and blending in between. That step is a record in the plant's own feed whose input fields are typed Swarm references to the batches it consumed, so lineage is a graph a reader walks backwards rather than a claim in a database, and the mass balance is arithmetic anyone can redo.

Prepaid, immutable storage (shipped). Storage is paid for upfront. A postage batch is bought with a capacity and a duration and funds the nodes holding those chunks for as long as it lasts, and buying it immutable is what makes a record unrewritable afterwards by anyone, its author included. This also names the real operational commitment: a provenance record has to outlive the product it describes, sometimes by years, and keeping a batch funded across that span is a job someone has to own rather than a property that arrives for free.

Anchoring time (external, optional). Signatures establish who, and content addressing establishes what, but neither establishes when. Writing a feed reference into a transaction on a public chain adds an independent witness that the record existed by a given block. The chain is external to Swarm, and this is the piece to add only if the timestamp is load-bearing, since it brings a second system's costs and failure modes with it.

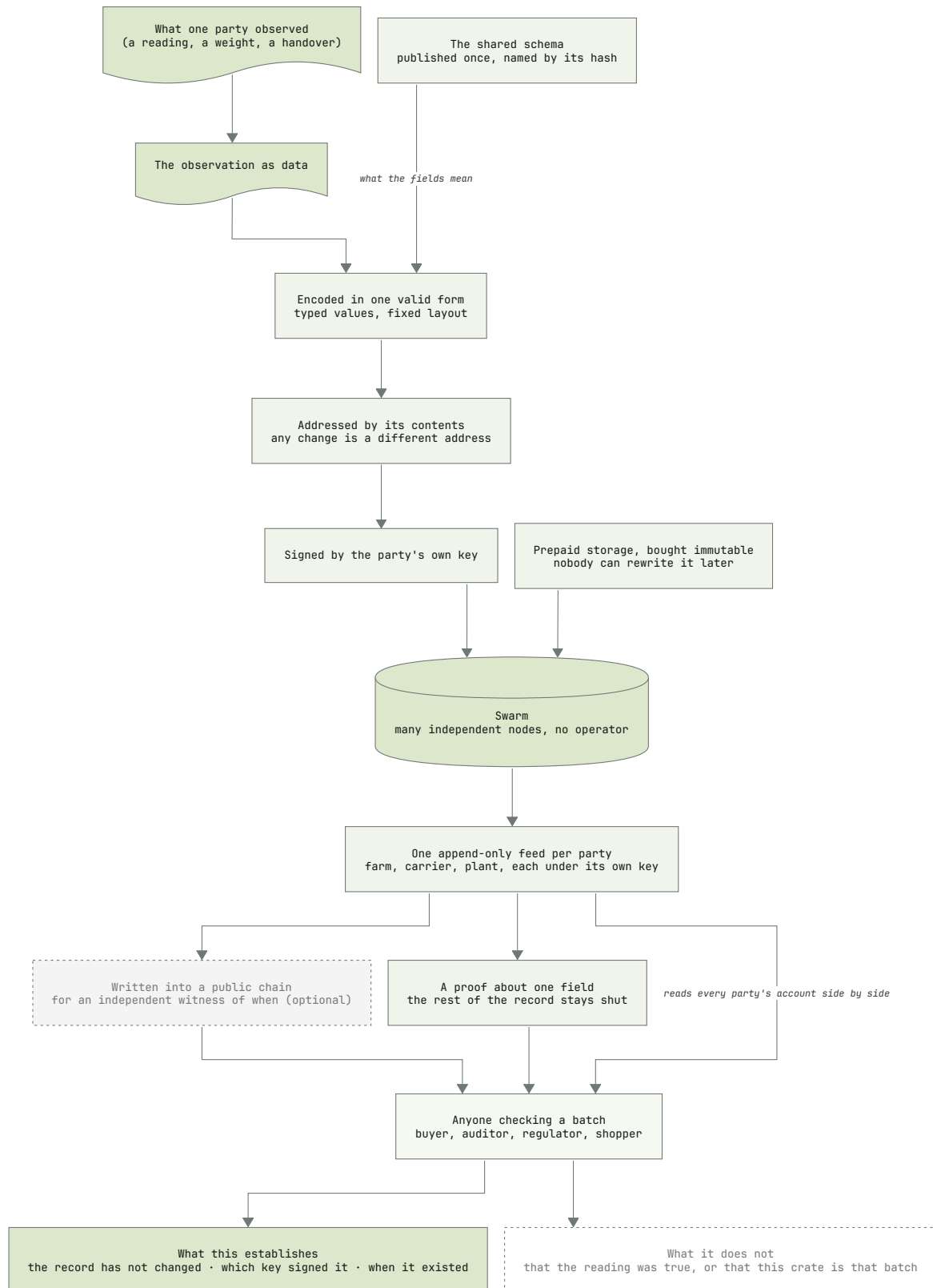
What none of this proves. The stack establishes three things: that a record has not changed since it was written, which key signed it, and, with an anchor, that it existed by a certain time. It establishes nothing about whether the record is true. A sensor can be miscalibrated, moved, or held next to a different pallet. A reading can be typed in by hand. An identifier on a crate can be transferred to a crate that never went through the field in question. This gap between the physical object and the digital record is where every provenance system either does careful work or quietly fails, and no storage format closes it. What the architecture gives is that each account is fixed when made, attributable to a named key, answerable in pieces, and readable beside accounts from parties with different incentives. Contradictions become visible and consistency becomes a track record. Copy that calls either of those proof should be rewritten.

On BeeSon's maturity. BeeSon is published, MIT-licensed and working, and it is version 0.1.1 with its last commit in October 2022 and a changelog that describes itself as an initial design. It is a Fair Data Society library rather than part of Bee, so it sits in Swarm's ecosystem rather than among its primitives. The format solves a problem that this space genuinely has and nothing else in the ecosystem addresses, and anyone adopting it is adopting an early library that has been still for a while. Both halves of that are true, and the second half is why this space is open.

PRIMITIVES Content addressing Feeds / Single Owner Chunks Postage stamps

SWARM ECOSYSTEM BeeSon bmt-js

EXTERNAL DEPENDENCIES GS1 EPCIS Chain anchor Field hardware



Data flow. What one party observed is encoded in one valid form against a shared schema that is itself published and named by its hash, addressed by its contents, and signed by that party's own key. It lands in prepaid immutable storage and is anchored in that party's own append-only feed, one per farm, carrier and plant. A reader sees every party's account side by side; a proof answers one field without opening the rest; an optional chain anchor witnesses when. The last two boxes separate what this establishes from what it does not.

Development

The write path is short. Build the record as a BeeSon value, serialize it, and put the bytes on Swarm; the address the network returns is the same address the sender computed locally, because the serialization is canonical.

```
const { BeeSon, TypeManager } = require('@fairdatasociety/beeson')

const record = new BeeSon({ json: { batch: 'C0-BB-2411', tempC: 3.8, massKg: 412 } })
const bytes = record.serialize()
const ref = await bee.uploadData(stamp, bytes)

const writer = bee.makeFeedWriter(Topic.fromString(batchId), signer)
await writer.uploadPayload(stamp, ref, { index })
```

The schema is the shared object. Define it once with `TypeManager`, keep it with `getDnaObject()`, and rebuild it with `TypeManager.loadDnaObject()` for every subsequent record so the whole series shares one type specification. Turning on `superBeeSon` moves that specification out of the records and behind a hash; `typeManager.superBeeSonAttributes()` returns the schema bytes and the Swarm address to publish them under, and every party writing against that schema references the same address. Deserialization becomes async because the reference has to be resolved, which `Utils.createStorage()` handles. Each participating party runs this side for itself under its own key, so nobody writes on anyone else's behalf.

Write the proof layer yourself. This is the honest shape of the work and it should be planned for rather than discovered. BeeSon lays bytes out so that inclusion proofs over individual fields are cheap; it does not generate them. That is `bmt-js`, which BeeSon already depends on for addressing. Nor does BeeSon ship an on-chain verifier: the phrase “verifiable cheaply on-chain” describes what the layout makes possible, and the contract that checks a proof against a stored address is something a team building this writes. Budget for it, and prototype the gas cost early, because the argument for the whole approach rests on that number being small.

Decide where readings get signed, before anything else. A reading signed by a device key on the device and a reading signed by a server after it has passed through a collection pipeline are different claims, and no amount of downstream cryptography can tell them apart later. If sensors post unsigned readings over HTTP to something that encodes and signs on their behalf, then every signature in the system attests to what that collector received. That may be an acceptable trade, and it has to be a decision rather than an accident. Field hardware, its gateways and any device key management are external to Swarm.

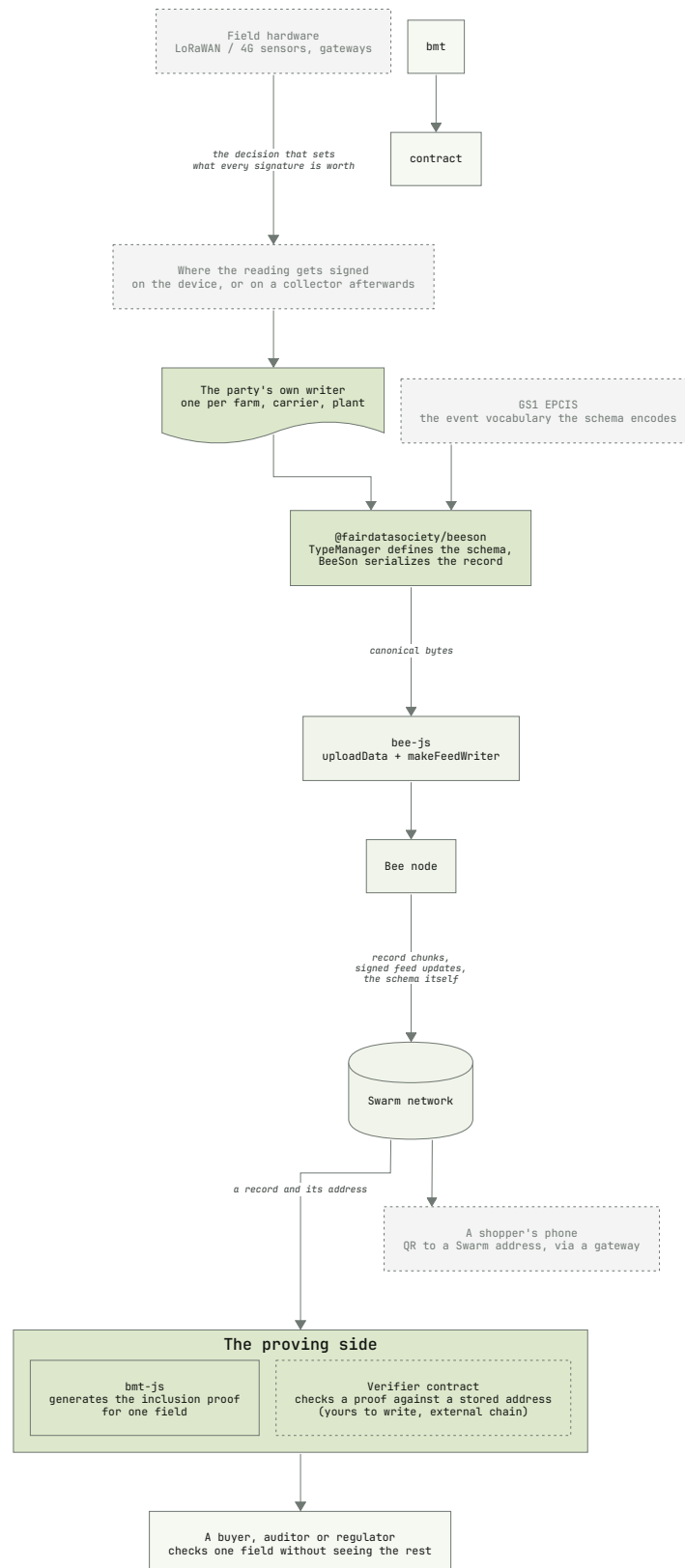
Do not invent the vocabulary. GS1's EPCIS already models what a provenance event has to answer, what object, what happened, when, and where, including the aggregation and transformation events this space depends on. Encoding EPCIS-shaped events as a BeeSon schema, rather than designing field names from scratch, is what makes the output legible to anyone who has integrated a supply chain system before, and it gives the shared schema its content. EPCIS is external to Swarm.

The consumer end. A QR encoding a Swarm address, or an ENS name resolving to a feed, gets a phone to a record without an app. Be straight about what that leans on: a phone browser reaches Swarm through a gateway, so resolution is the one step in the flow a reader cannot check for themselves, and a build that takes this seriously lets a determined reader point at their own node instead. Keep resolvers, indexes and image hosting out of the trust story. They are convenience

infrastructure, and a provenance system whose photographs live on a service that can withdraw them has moved the problem rather than solved it.

TOOLS & REFERENCES

- BeeSon — <https://github.com/fairDataSociety/beeson> (@fairdatasociety/beeson, MIT, v0.1.1)
- bmt-js — <https://github.com/fairDataSociety/bmt-js> (Binary Merkle Tree, inclusion proofs)
- swarm-cid-js — <https://github.com/ethersphere/swarm-cid-js>
- bee-js — <https://bee-js.ethswarm.org>
- Bee Feeds documentation — <https://docs.ethswarm.org/docs/develop/tools-and-features/feeds>
- Postage stamps — <https://docs.ethswarm.org/docs/concepts/incentives/postage-stamps>
- swarm-cli — <https://github.com/ethersphere/swarm-cli>
- **Swarm ecosystem, not Bee primitives:** BeeSon and bmt-js are Fair Data Society libraries
- **External dependencies (not Swarm):** GS1 EPCIS <https://www.gs1.org/standards/epcis> a public chain, if a timestamp anchor or an enforced transformation is needed; field hardware and its key management



Component assembly. Field hardware feeds a writer that each party runs for itself; where the reading gets signed, on the device or on a collector afterwards, sets what every signature downstream is worth. BeeSon serializes against a schema encoding an EPCIS-shaped vocabulary, bee-js uploads the canonical bytes and writes the feed, and the record, the feed update and the schema all live on Swarm. The proving side is the part a team builds: bmt-js generates an inclusion proof for one field and a verifier contract checks it against the stored address.

Reference implementations

Yours to take. The Swarm primitives are shipped and documented, the record format exists as a working reference implementation, and no production system has been built on the combination that could be verified.

BeeSon (Fair Data Society, author Viktor Levente Tóth) — <https://github.com/fairDataSociety/beeson> — the format described throughout this document, MIT, published as @fairdatasociety/beeson, Node and browser with TypeScript definitions. Canonical typed serialization, segment alignment for cheap BMT inclusion proofs, and SuperBeeSon schema references. Its tests cover integers, floats, strings, booleans, BigInt, manifest and feed CIDs, typed arrays, nullable containers and SuperBeeSon round-trips, which is enough to show the format works. Version 0.1.1, last commit October 2022, self-described as an initial design. Treat it as a designed format with a reference implementation, not as maintained infrastructure.

BDF Protocol (Marcio Barradas, Antonio Gonzalo, Berto Rial) — Swarm accelerator entry, August 2026. The Bio-Digital Footprint protocol structures IoT field data from blueberry deployments in Colombia and kiwi in Galicia, plus freeze-drying plant telemetry, into typed Raw Product and Processed Product records: BeeSon-encoded, BMT content-addressed, signed as feed updates, with the feed reference anchored on Sepolia for block timestamping and a QR resolving the chain. A prototype, and labelled as one here. At the time of writing there is no public repository, the pipeline is stated as the goal of the event rather than as a working system, and the anchor is on a testnet. Two things make it the right thing to point at anyway: it has real field deployments behind it rather than synthetic data, and it takes on the transformation event, which is the part of a food supply chain that most provenance demonstrations avoid. Its own architecture diagram claims integrity, authenticity and timestamp, and stops there, which is the correct set of claims.