Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

PointCloud-ID: Deterministic Cross-Format Identity and Attribute-Aware Integrity Verification for Geospatial Point Clouds

A. Urias

alex@aurtech.mx

Independent Researcher

Method Article

Keywords: point cloud, LiDAR, content identity, data integrity, provenance, C2PA

Posted Date: August 12th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10490225/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

PointCloud-ID: Deterministic Cross-Format Identity and Attribute-Aware Integrity Verification for Geospatial Point Clouds

A. Urias[1*]

[1*]Independent Researcher, Hermosillo, Sonora, Mexico.

Corresponding author(s). E-mail(s): alex@aurtech.mx
ORCID: 0009-0007-3147-323X;

Abstract

A geospatial point cloud has no stable identity: the same survey re-encoded from LAS to LAZ to COPC yields different bytes and different file hashes, leaving a verifier unable to distinguish a content-preserving re-encoding from a modified copy. We present PointCloud-ID, a deterministic content identity that survives re-encodings preserving a declared grid, and that detects and spatially localizes changes to in-scope geometry or attributes. The construction has three stages. First, canonicalization under a declared, versioned profile: coordinates quantize to the profile's grid following LAS scale/offset semantics, the coordinate reference system (CRS) is fixed, the multiset is ordered by Morton code, and the attribute scope is enumerated. Second, an attribute-partitioned Merkle tree over spatially chunked canonical data yields per-attribute roots, folded with a canonical digest of the profile into one identity. Because that digest enters the identity itself rather than an accompanying assertion, identical records declared under a different CRS or profile version cannot share an identity; changes localize to the affected chunks and attribute domains with logarithmic membership proofs. Third, a signed manifest (RFC 8785 canonical JSON, COSE/Ed25519) records tool provenance and maps onto a namespaced C2PA assertion. Lossy transforms intentionally break the identity and are recorded as explicit derived-from links. The scheme attests that signed canonical content is unchanged, not that the signed scene reflects reality. The accompanying artifact provides the profile specification, reference vectors, implementation, fixed experimental inputs, and deterministic evaluation protocol. Identities and test-vector outputs reproduce exactly, while timing and memory measurements remain environment-dependent.

Keywords: point cloud, LiDAR, content identity, data integrity, provenance, C2PA

1 Introduction

A contractor finishes a highway embankment and delivers a drone LiDAR survey to the project owner as evidence of the as-built earthwork volumes. Two years later a drainage failure triggers an insurance dispute. The owner produces a COPC file; the contractor's archive holds the original LAS; the insurer's consultant was given a LAZ copy by email. All three parties believe they hold "the survey." The files' SHA-256 digests all differ, as they must, because LAS, LAZ, and COPC encode the same points in different bytes; file hashes cannot say whether the three copies agree. Neither can visual inspection: a shifted embankment toe, a re-classified ground surface, or a deleted low region changes volumes materially while looking plausible. Today the dispute turns on custody testimony about files that no one can technically tie to one another.

That dispute is the sharp case. The ordinary one is more common and just as unresolved: we often hold the same survey as LAS, as LAZ, as COPC, and again after some tool has rewritten it, and no byte hash can confirm that these are the same data. A file hash answers a question about one encoding, not about the survey. So the everyday question — do I already have this? — has no mechanical answer, and deduplicating an archive, confirming that a delivery matches what was sent, or checking that a conversion changed nothing all fall back on filenames and trust.

The same gap recurs wherever LiDAR is treated as evidence and circulates across tools and parties (Destefanis et al. 2025). In *insurance*, before- and after-event aerial scans of a flood or wildfire site pass among insurer, adjuster, and contractor; a content identity ties those copies and can flag a re-classified damage extent while the geometry is untouched. In *carbon and forestry MRV* (measurement, reporting, verification), the biomass estimates that back carbon credits rest on airborne LiDAR; a verifiable identity lets an auditor confirm the cloud behind a credit is the one that was signed and detects a swapped or precision-reduced tile — integrity of the input, not correctness of the estimate. In *cultural-heritage and scientific archives*, a canonical scan re-encoded across decades of format migrations keeps one citable identity and surfaces silent corruption. In each case the useful property is not a claim about the world but a verifiable statement that *this data is what was recorded and signed*.

The missing primitive is an *identity* for the point cloud itself, rather than for any one encoding of it. We argue such an identity must satisfy three properties simultaneously: it must be deterministic (any two conforming implementations agree bit-for-bit), re-encoding-invariant (LAS→LAZ→COPC round trips at unchanged precision preserve it), and fragile against any change to the profile's in-scope canonical representation (any such change breaks it, and the verifier learns *what kind* of change, and *where*).

PointCloud-ID provides this (Fig. 1). Identity exists only *under a declared canonicalization profile*. We make no claim of a universal cross-format hash, which we believe is not well-defined: formats disagree about attribute precision, and any fixed precision choice is itself a profile. Cross-format equality is instead a *verified property of a shared profile* — two files may carry the same identity exactly when both declare the same profile and version and both canonicalize to the same integer records, a fact a verifier checks rather than a promise the format makes. A re-encoding that preserves the declared profile's grid and every in-scope attribute preserves the ID. Transformations

that change the canonical content — such as downsampling, reprojection, precision reduction, or attribute stripping — produce a new identity and may be recorded through derived-from provenance links. The scheme attests that data is unchanged since signing; it cannot attest that the signed scene reflects reality (Sect. 3).

Contributions.

1. A versioned **canonicalization-profile specification** with normative quantization, ordering, serialization, and tie-breaking rules, published with reference test vectors.
2. An **attribute-partitioned Merkle identity construction** giving deterministic identity, per-attribute tamper localization, and $O(\log n_{\text{chunks}})$ membership proofs, over a **profile-bound identity**: a canonical digest of the declared profile is folded into the identity, so agreement on an ID already implies agreement on the profile and CRS that produced it.
3. A **C2PA bridge**: a concrete mapping and validated proof-of-concept linking PointCloud-ID manifests to C2PA assertions.
4. A **reproducible evaluation suite**: a deterministic E1–E5 protocol with fixed seeds, fetch scripts, and pinned tools.

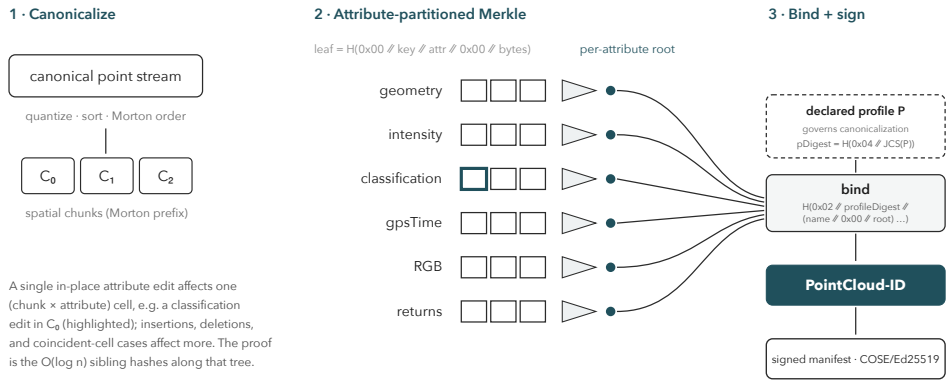


Fig. 1 The construction. Canonical points are spatially chunked; each attribute channel is hashed into its own Merkle tree over the chunks; the per-attribute roots are folded together with a digest of the declared profile to give the PointCloud-ID, which the manifest then signs. A single in-place edit localizes to one (chunk × attribute) cell; insertions, deletions, and coincident-cell cases affect more, each with an $O(\log n_{\text{chunks}})$ membership proof

2 Background and Related Work

2.1 Point-cloud formats and why file hashes fail

LAS stores coordinates as 32-bit integers under per-file scale/offset; the doubles users see are reconstructed (ASPRS 2019). LAZ compresses LAS losslessly (rapidlasso

Table 1 Watermarking versus PointCloud-ID

	3D watermarking	**PointCloud-ID**
Mechanism	embeds a signal *into* the data	digest *of* the data; data untouched
Modifies data	yes	none
Robustness	survive edits	fragile by design
Localizes	no	(chunk × attr.)
Precision	statistical	exact, cryptographic

GmbH 2025; Open Geospatial Consortium 2026); COPC is a LAZ 1.4 file whose points are clustered into an octree of LAZ chunks for streaming (Hobu, Inc. 2021). A lossless LAS→LAZ→COPC round trip preserves every point's integer coordinates and attributes but permutes point order and rewrites every container byte. File digests therefore differ across encodings of identical content, and, worse for forensics, also differ across *tools* producing the same encoding. One gap the profile must close: LAS specifies how integers reconstruct to coordinates but leaves the *forward* rounding (double to integer) unspecified, so even "the same precision" is not one rule in the wild.

2.2 Watermarking is a different problem

3D watermarking *embeds* a signal into geometry or attributes so that it survives modification; current schemes span deep-neural embedding (Zaman et al. 2025), robust copy detection (Yang et al. 2022), and feature-point constructions using ISS keypoints and Mahalanobis distances (Zhang et al. 2024). PointCloud-ID is the complement, not a competitor (Table 1). A plain hash of sorted records remains insufficient: it yields one whole-cloud verdict, without format-aware canonicalization, attribute-level localization, membership proofs, or signed profile governance. The contribution over `SHA-256(sort(points))` is (a) format-semantics-aware canonicalization that makes the digest *well-defined* across encodings; (b) attribute partitioning, so "classification edited, geometry intact" is a verifiable statement (with a known exception at coincident cells, Sect. 7); (c) localization proofs; (d) signed, versioned profile governance.

2.3 Merkle trees, multiset hashing, prior systems

Merkle trees (Merkle 1988) give logarithmic membership proofs and are used for image tamper localization by hashing tiles into a hierarchy (Chen et al. 2019); Certificate Transparency (Laurie et al. 2021) standardizes the domain-separated construction we adopt. An instructive alternative primitive is the incremental multiset hash (Clarke et al. 2003; Maitin-Shepard et al. 2017): it hashes an unordered multiset directly, but it cannot localize a change to a region, cannot attribute it to a channel, and admits no logarithmic proofs; cryptographic accumulators (Ozcelik et al. 2021) share the limitation. The closest prior system securing point clouds, SAUSA (Xu et al. 2022), uses blockchain-mediated access control, which protects *who may touch* the data but gives honest parties no content identity once copies circulate. To make this concrete we ran

Table 2 Content-hash schemes on the properties the dispute workflow needs. *Channel* is per-channel (attribute) localization; *Path hashes* counts the hashes a verifier must see to check one region (whole = the entire cloud). The sort-based schemes share one $O(n \log n)$ canonicalization and the rest are $O(n)$

Scheme	X-fmt	Reord.	Reg.	Channel	Path hashes
SHA-256 of file bytes	**no**	no	no	no	whole
SHA-256 of sorted records	yes	yes	no	no	whole
Chunk Merkle, attrs merged	yes	yes	yes	no	2
Additive multiset hash	yes	yes	no	no	whole
PointCloud-ID	yes	yes	yes	**yes**	**2**

all four alternatives and PointCloud-ID over one 200 000-point cloud (Table 2, author-generated synthetic data, four chunks, equally optimized). A file hash is not even cross-format invariant. Canonicalizing before hashing recovers invariance but gives one whole-cloud verdict. Chunking adds region localization and an $O(\log n)$ proof yet, hashing a chunk's attributes together, cannot say *which* channel changed. The multiset hash trades all localization for a no-sort pass, at one digest per record. Only the attribute-partitioned tree gives region *and* channel localization with a two-hash proof; the extra forensic resolution costs about 200 manifest bytes. Two proof notions differ: the Merkle path needs $O(\log n)$ sibling hashes, while a fully self-contained proof (verifiable without the signed manifest) additionally carries the per-attribute roots together with the profile digest the identity folds ahead of them (a constant, seven 32-byte digests for profile v1); a proof checked against an already-available signed manifest needs only the path. The $O(\log n)$ figure counts sibling authentication hashes, logarithmic in the number of nonempty chunks in the relevant attribute tree. Establishing that a named chunk holds particular canonical content additionally requires the target leaf's preimage. The verifier recomputes the leaf from the chunk key, attribute name, and canonical payload. When that payload is bundled with the proof, only the authentication-path overhead remains logarithmic in the number of nonempty chunks.

2.4 C2PA and content addressing

C2PA binds assertions about an asset's origin to a cryptographic manifest (Coalition for Content Provenance and Authenticity 2026). Published critiques document trust-list governance and the gap between provenance and truth: a signer can sign a fabricated asset (Golaszewski et al. 2026; Nemecek et al. 2026). The digital-preservation community faces the same gap from the archival side: OAIS fixity is a checksum over one encoding, so every format migration invalidates it (Giaretta 2011). git and IPFS (Benet 2014) name objects by digest, which is identity for *bytes*, not for *content under re-encoding*. To the author's knowledge, no prior work provides deterministic cross-format identity for geospatial point clouds; the claim rests on the literature search conducted for this paper.

Table 3 Adversary capabilities and outcomes

Attack	Detected	Localized
Modify coordinates ≥ 1 grid step	yes	chunk(s)
Modify one attribute value	yes	chunk × attr.
Delete or insert points	yes	chunk(s)
Reorder points	no effect	—
Re-encode preserving declared canonical content	no effect	—
Strip an in-scope attribute	yes	attr. root
Splice regions from two clouds	yes	chunk(s)
Replay an old signed manifest	**no**	—
Move data below one grid step	**no**	—
Forge a scene, sign with a valid key	**no**	—
Steal or coerce the signing key	**no**	—
Re-declare under a different profile or CRS	**yes**	different identity, not localized

3 Threat Model

Assets: a point cloud, its manifest, and the signing key. **Parties:** producer (signs), verifier (checks), adversary. Table 3 summarizes attacker capabilities and outcomes. **Trust assumptions:** SHA-256 collision resistance and Ed25519 unforgeability under their standard assumptions; signing-key custody; a trust list mapping keys to real-world signers. We do not assume trusted time; freshness, if needed, comes from external timestamping, named as future work.

Security argument. Let $C_P(X)$ denote the canonical representation of cloud X under profile P. Under the collision and second-preimage resistance of SHA-256, if two clouds differ in any in-scope leaf preimage under the same profile, the affected attribute root and final PointCloud-ID differ except with negligible probability. If the declared profile differs, its canonical profile digest and therefore the final identity likewise differ under the same assumption. Ed25519 unforgeability prevents an adversary without the signing key from replacing the signed manifest. These arguments provide neither freshness, signer trust, nor assurance that the signed scene reflects reality; the replay, key-custody, sub-grid, and truthfulness limitations in Table 3 stand unchanged.

4 The PointCloud-ID Construction

Profile v1 is the profile evaluated in this article; it is the methodology whose properties the E1–E5 experiments establish. Profile v2 is a separately implemented, association-sensitive revision that removes the coincident-cell contamination discussed in Section 7; it is not put through the full E1–E5 protocol here, and its broader validation is future work. This section states the v1 construction and its costs (Fig. 1). Coordinates quantize to $q \in [0, 2^{21})$ per axis; Morton codes are 63-bit integers; all digests are SHA-256 with domain-separation prefixes.

Grid domain and what invariance means. The identity is defined on the profile's grid, and that grid is deliberately coarser than the LAS int32 coordinate domain: canonicalization is a projection onto 2^{21} steps per axis, not a lossless encoding of the

input coordinates. Two consequences follow, and we state both as scope conditions rather than leaving them implicit. First, invariance holds exactly for re-encodings that preserve the declared grid; a writer that renegotiates scale or offset does not by itself change anything, because canonicalization reads the *declared* profile, never the container's header: what matters is the decoded coordinates and the profile they are quantized under. Invariance therefore holds for any re-encoding that preserves the decoded coordinates relative to the declared grid — which is what E1 measures across LAS, LAZ, and COPC. A re-encoding produces a different identity when it changes those decoded coordinates, when it declares a different profile, or when its coordinates cannot be placed on that grid unambiguously. Changing container metadata alone, with the decoded coordinates and the declared profile intact, leaves the identity untouched. This is why the profile is declared and checked rather than inferred from the file. Second, inputs differing only below one grid step are the same canonical cloud and share an identity by construction; sub-grid perturbation is outside what the scheme detects (Section 7). A dataset whose extent divided by the declared scale exceeds 2^{21} steps on any axis is therefore not representable under that profile: the reference implementation computes the required steps per axis from the header extent and rejects the profile before reading any point, rather than wrapping, clamping, or emitting an identity for a truncated domain. Surveys needing finer resolution over a wide extent are served by a coarser scale, by tiling, or by a future profile with a wider code.

The construction has three stages; Algorithms 1 and 2 give the first two.

Ties at quantization boundaries are ties of the computed binary64 value, not of decimal notation; the quantization is deterministic under IEEE-754 binary64 evaluation with a fixed operation order and fused multiply-add disabled. Duplicates are preserved as multiset members. Reordering attacks vanish here: canonical order is a pure function of content.

Binding the chunk key into each leaf means a chunk cannot be transplanted between clouds without breaking its leaf; this turns splicing from "plausible" into "localized detection." The tree is the RFC 9162 (Certificate Transparency) Merkle Tree Hash (Laurie et al. 2021): the $0x00/0x01$ prefixes are that standard's, and the bottom-up fold we implement — pair adjacent nodes, promote an unpaired node unchanged — produces the same tree as the RFC's recursive split-at-the-largest-power-of-two definition. We check this by test against an independent transcription of the RFC's recursion for every leaf count from 1 to 160. Promoting the unpaired node is not the duplicate-last rule, which admits the distinct-trees-same-root ambiguity of CVE-2012-2459. Identity chunks are content-defined, independent of any file's *storage* chunking; when the input already carries integer coordinates on the profile's exact grid, implementations take those integers as-is (passthrough is provably equivalent to reconstruct-then-requantize).

Algorithm 1: Canonicalization $(O(n \log n))$

Input : point set; profile P (scale, offset, chunkDepth, attributes)
Output: spatial chunks in canonical order
foreach *point* p **do**
  $q_a \leftarrow \left\lfloor (p_a - P.\text{offset}_a)/P.\text{scale}_a + \frac{1}{2} \right\rfloor,\ a \in \{x, y, z\}$; `// IEEE-754; no FMA`
  assert $0 \le q_a < 2^{21}$;
  $m(p) \leftarrow \text{MORTON63}(q_x, q_y, q_z)$;
sort points by (m, int, class, gps, rgb, ret);
chunks $\leftarrow$ group by the top $3\,P.\text{chunkDepth}$ bits of m;
return chunks;

Algorithm 2: Attribute-partitioned identity $(O(n))$

Input : chunks in canonical order; profile P
Output: the PointCloud-ID
foreach *attribute* $a \in P.\text{attributes}$ **do**
  foreach *chunk* c *(ascending key* k*)* **do**
    $\text{leaf}(c, a) \leftarrow H(\texttt{0x00} \,\|\, k \,\|\, a \,\|\, \texttt{0x00} \,\|\, \text{SERIALIZE}(c, a))$;
  $\text{root}[a] \leftarrow \text{MERKLEFOLD}$; `//` $H(\texttt{0x01} \,\|\, L \,\|\, R)$
ID $\leftarrow$

$$H\left(\texttt{0x02} \,\|\, \text{PROFILEDIGEST}(P) \,\Big\|\, \Big\|_{a \,\in\, P.\text{attributes}} [\text{UTF8}(a) \,\|\, \texttt{0x00} \,\|\, \text{root}[a]]\right);$$

return ID;

What the identity binds. The large $\|$ in Algorithm 2 concatenates over the attributes *in the profile's declared enumeration order*, one $\text{UTF8}(a) \,\|\, \texttt{0x00} \,\|\, \text{root}[a]$ block each; the order is part of the profile, so it is fixed before any point is read. The identity folds a canonical *profile descriptor* alongside the per-attribute roots, so every declared field (version, CRS, scale, offset, chunk depth, attribute scope) reaches the digest: $\text{PROFILEDIGEST}(P) = H(\texttt{0x04} \,\|\, \text{JCS}(P))$, where JCS is RFC 8785 canonical JSON (Rundgren et al. 2020) applied to the profile object with its natural JSON types (numbers as numbers, arrays as arrays). Serialization is inherited from the standard rather than defined by this paper, so any conforming RFC 8785 implementation reproduces the digest; a reimplementer needs no numeric convention of ours. Two clouds whose integer records coincide but whose declared CRS differs therefore receive different identities, and a bare digest comparison is meaningful on its own. The attribute name likewise enters each leaf preimage, so a leaf cannot be replayed across channels. Profile v1.0.0 bound these fields only in the signed manifest, which left a bare digest blind to them; v1.1.2 moves the binding into the digest and supersedes it. Because the profile is part of the preimage, a profile mismatch is detected as a different identity rather than silently compared.

Reading the version numbers. Two independent things are versioned. A *line* (profile v1, profile v2) names a construction; a *profile version* inside a line (1.0.0, 1.1.2, 2.0.2) names a released set of rules, and because the version string is folded into the

Table 4 Version identifiers, what changed, and what covers each

Identifier	What changed	Vectors	Experiments
profile v1	the construction of this article	—	E1–E5
1.0.0	first release; profile bound only in the manifest	superseded	—
1.1.2	profile digest folded into the identity; attribute name in each leaf; RFC 8785 numbers and arrays	`toy-10pt`, `toy-proof`, `toy-signed`	E1–E5 as reported here
profile v2	association-sensitive revision (ADR 0002)	—	contamination
2.0.2	per-attribute projection roots + record root	`toy-v2-10pt`	only

digest, every release defines its own identities. So v2 is a different construction, not "version 2" of the numbering. Table 4 maps each identifier to what it owns.

Stage 3 — sign. The manifest {id, profileVersion, crs, attributeRoots, method, tool} serializes as RFC 8785 canonical JSON and is signed as COSE_Sign1 (Schaad 2022b) with Ed25519 (Schaad 2022a). **Verify** is recompute-and-compare: $O(n \log n)$, no network, no service. **Prove** (chunk × attribute) membership: the Merkle path to that attribute's root plus the other roots, $O(\log n_{\text{chunks}})$ hashes.

5 C2PA Bridge

The manifest is transport-agnostic; C2PA is the carrier with the most to offer, because verifying tooling and trust-list governance already exist there. Under C2PA (the 2.x line), hashing algorithms for the standard binding assertions are allow-listed, and an unlisted algorithm is forbidden, so the PointCloud-ID must not masquerade as `c2pa.hash.data`. It travels instead as a namespaced *custom assertion*, which the specification places outside that algorithm governance (Table 5, Fig. 2); that carrier is a stable part of the 2.x line (from 2.1 through 2.4), so the mapping does not depend on the exact minor version. Carrying both bindings turns the desynchronization critique (Nemecek et al. 2026) into a diagnostic: the byte hash broken with the PointCloud-ID intact establishes *profile-scoped content equivalence* despite a representation change — consistent with a content-preserving re-encoding, though it does not by itself prove the transformation path; both broken means *content change*. The bridge does not strengthen C2PA's trust model; the documented weaknesses pass through unchanged. The mapping is exercised, not just described: a proof-of-concept (`eval/c2pa/build_c2pa_poc.sh`) emits a real manifest carrying the toy PointCloud-ID as the custom assertion `org.pointcloud-id.identity`, signs a carrier asset, and validates it with `c2patool` 0.27.2 (`c2pa-rs` 0.90.2; the exact binary is pinned by recorded SHA-256 in the artifact) to `validation_state: Valid` with the assertion intact. The exact tool version is not load-bearing: C2PA 2.x implementations that preserve and expose namespaced custom assertions can carry this manifest; the proof of concept validates the pinned implementation reported here. Three layers stay distinct: structural validity (the assertion round-trips), signature validity (COSE_Sign1 verifies), and signer trust (whether the certificate chains to an accepted anchor). The PoC

Table 5 Manifest fields mapped onto C2PA 2.x carriers

Manifest field	C2PA carrier
id, profileVersion, crs, attributeRoots	custom assertion `org.pointcloud-id.identity`
tool	`claim_generator_info`
Ed25519 signature	claim signature (COSE)
encoding bytes of *this* file	`c2pa.hash.data` (kept alongside)
derived-from link	`c2pa.actions` + `c2pa.ingredient`

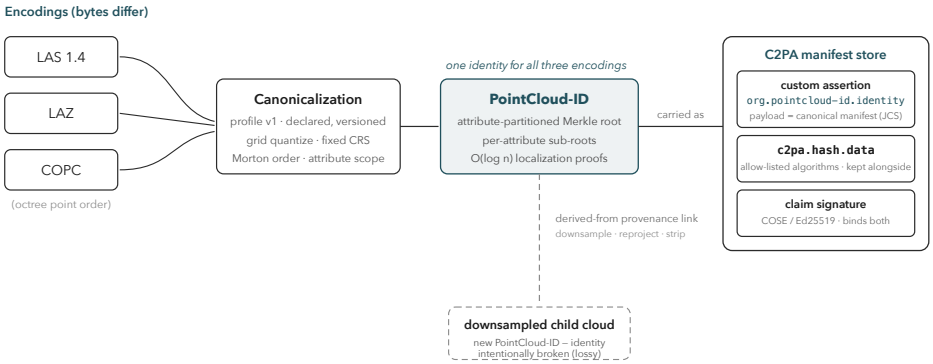


Fig. 2 Lossless re-encodings converge, under a declared canonicalization profile, on one PointCloud-ID, carried as a namespaced C2PA custom assertion beside the standard hard binding; lossy derivations break identity into an explicit derived-from link

establishes the first two; signer trust is C2PA's trust-list concern, and the throwaway test chain is deliberately untrusted.

6 Evaluation

Every reported result is generated by the committed evaluation protocol: pinned datasets, a fetch script, pinned tool versions, fixed seeds, and medians over at least five runs. Identities and test vectors reproduce exactly; timing and memory figures are hardware-dependent and are reported with their observed ranges. Controlled experiments use author-generated synthetic data (seeded clouds and a realistic generator with tunable coincident-cell density); E1, E2, E4, and E5 additionally run on a real public cloud (Autzen, 10,653,336 points, COPC 1.0) (Watershed Sciences, Inc. and Sampson 2021) via the separately implemented NumPy recomputation.

E1 — invariance. A 20,000-point cloud written as LAS, LAZ, and converted to COPC and back with PDAL 2.10, all read back and hashed, yields one identical ID across all five representations. On the real Autzen cloud, a separately implemented

Table 6 E1 invariance: all representations yield one ID

Representation	PointCloud-ID (first 16 hex)
direct (pre-encode)	`c2722a9618cb045f`
LAS → decode	`c2722a9618cb045f`
LAZ → decode	`c2722a9618cb045f`
LAZ → COPC → decode	`c2722a9618cb045f`
COPC → LAS → decode	`c2722a9618cb045f`

Table 7 E2 tamper sensitivity and localization

Tamper	Detected	(chunk×attr) pairs
move one point +1 grid step	yes	1
flip one classification code	yes	1
alter one intensity value	yes	1
delete one point	yes	6
splice 500 foreign points	yes	90
reorder points only	no change	0

NumPy recomputation confirms the same ID across LAS/LAZ/COPC and a round trip. Container choice does not affect the identity (Table 6). Two independent LAZ implementations — lazrs (Rust) and LASzip (C++) — are exercised on both sides: each reader yields the same ID on the real LAZ and COPC, and re-encoding the points with each writer then decoding with each reader reproduces it in all four encoder × decoder combinations. Identity is independent of the LAZ tool, not just the container.

E2 — sensitivity + localization. On a 20,000-point cloud (16 chunks), each injected tamper is detected, reorder-only is correctly ignored, and every membership proof (4 hashes = $\log_2 16$) verifies (Table 7). An in-place attribute edit isolates to one pair; deletion or splice touches all channels of the affected chunks. The same matrix on the full Autzen cloud reproduces every outcome: the four single-attribute edits (geometry, classification, intensity, and a 1-ULP GPS-time change) each localize to exactly one (chunk × attribute) pair, with none of the coincident-cell perturbation the synthetic worst case constructs — the v1 contamination condition was not observed on the real, clustered LiDAR we evaluated, and v2 removes it.

E3 — boundary + determinism. The same cloud yields an identical ID across runs; duplicates are preserved with a stable ID; reorder-invariance is confirmed. The committed vector was reproduced in all six tested software environments for profile 1.1.2 by one harness (`eval/e3_platforms.sh`) that writes each row from the command's own output and compares against the committed vector afterwards, rather than recording an expected value: six environments, all reproducing `526048e1...` (`eval/results/e3-platforms.json`). They are macOS/arm64 and macOS/x86_64 on Python 3.9.6, macOS/arm64 on Python 3.14.6, Linux/arm64 and Linux/x86_64 on Python 3.12.13, and the NumPy second stack, which reproduces both the v1 and

Table 8 E4 reference-implementation cost (median of 5 runs, with observed min–max for both columns)

Points	ID time (ms)	range	peak RSS (MB)	range
10^4	30.4	29.1–32.2	251	251–251
10^5	397.9	384.2–417.1	267	254–324
10^6	5377.5	4652.7–6151.1	1034	734–1334

v2 toy vectors together with the Morton and number-rendering known answers. The macOS x86_64 run is through Rosetta 2 and the Linux runs are containers on the same Apple-silicon host, x86_64 under emulation; they exercise different arithmetic code generation and a second operating system, not independent physical hardware. Several environments therefore used emulation on the same Apple-silicon host, so these results provide cross-environment evidence rather than a universal platform proof. The committed workflow also runs on GitHub's Linux x86_64 runner. A committed runner reproduces both IDs and the grid-boundary ties.

E4 — performance. ID computation time and peak RSS versus point count, median of 5 runs, on an Apple M3 Max (Table 8). These establish feasibility; three sizes cannot measure a complexity exponent, and the reference's boxed-object path caps near 10^6 points, so the numbers are a reference baseline, not an optimized bound. The table reports the observed min–max alongside both medians, because neither quantity is stable to the precision a bare median suggests. Time varies by roughly 25 % across the five runs at 10^6 points, and peak resident set by rather more — 734 MB to 1334 MB for the same workload — since RSS is sampled after each run and so tracks when the collector happened to run rather than the algorithm's requirement. Read the memory column as an order of magnitude, not a measurement. Profile v1.1.2 adds one profile-digest hash per identity and one attribute name per leaf preimage, work bounded by chunks rather than points, so the curve is unchanged from earlier profile versions within that spread. Two typed-array paths remove that memory ceiling: a column-oriented path in the TypeScript reference itself (`impl/src/typed.ts`, byte-for-byte equivalent to the boxed path by test) computes a 10^7-point ID in about 1.4 GB; and the NumPy path gives the real-Autzen curve across four decades — 3 ms at 10^4, 0.27 s at 10^6, 4.4 s at the full 10^7 — at roughly 66 bytes/point above the fixed decode floor (629 MiB of additional resident set at 10^7; isolating the computation from that floor). All three agree on the ID at every size.

E5 — derived-from links. Two lossy transforms, each of which must break identity and be recorded rather than passed off as identity. Decimating the cloud 2:1 changes the ID; reprojecting EPSG:32614→32615 changes it again; in both cases the child manifest carries a derived-from block naming the parent ID and the action. On the real Autzen cloud both break the ID as well — a 2:1 decimation, and a genuine reprojection from NAD83 Oregon Lambert (ft) to UTM 10N (m) via pyproj — each recorded as a `downsampledFrom`/`reprojectedFrom` link rather than silently accepted.

7 Threats to Validity

Cross-attribute contamination among coincident points. Per-attribute isolation is not universal in the v1 construction (Fig. 3). Each channel is serialized in the single canonical order. When two points share a quantized cell and are tied on every sort key of higher priority than an edited attribute, editing it can flip their order and permute lower-priority channels, changing those roots even though their values did not. We confirmed this in the reference implementation. We measured it on generated clouds: with realistically distinct coincident returns, single-attribute edits contaminated another channel in 1 of 3,600 trials — one intensity edit at 5% coincident-cell density, with no contamination for classification, GPS-time, or return-number edits at any tested density — because the high-priority intensity channel usually separates the twins; in a constructed intensity-tied worst case, a classification edit changes the gpsTime root in 500 of 500 trials. A construction that removes the dependence on a single mutable global order — per-attribute projection roots (each channel serialized in the order induced by Morton and its own value) plus a full-record root — is specified in ADR 0002 and implemented; on the same worst-case data it reduces the contamination rate from 500/500 to 0/500 while still detecting the edit. The positioning is explicit, and we state the argument rather than leaving it to be inferred. Profile v1 is the profile evaluated in this article. Its contamination is not a latent defect of unknown reach: it has a stated trigger condition, and we measured how often that condition arises. Contamination requires two points in one quantized cell that are additionally *tied on every sort key of higher priority than the edited attribute* — in practice, tied on intensity. Returns that share a cell are usually separated by intensity or by GPS time, which is why the measured rate across 1%, 2%, and 5% coincident-cell density is 1 of 3,600 single-attribute-edit trials. The triggering condition was not observed in the evaluated Autzen cloud. The 500/500 figure comes from data constructed to force the tie. The honest summary is therefore a conditional one: v1's per-attribute isolation holds except where an adversary or an unusual sensor produces intensity-tied coincident returns, and a deployment that expects such data should declare v2 instead. We do not present v1 as unconditionally isolating, and we do not present v2 as validated: v2 is implemented and shown to eliminate the worst case on the same data (500/500 $\rightarrow$ 0/500), but its full E1–E5 evaluation is future work, so this article evaluates and commits vectors for v1 only. Figure 3 contrasts the two.

Duplicate-geometry association ambiguity. Because a point carries no persistent identifier, when several points occupy the same quantized cell the scheme cannot always distinguish an attribute edit on one of them from a permutation of associations within the cell. This is fundamental to any identity over an unlabeled multiset.

Other threats. Quantization ties are defined on the computed binary64 value, deterministic across IEEE-754 platforms provided fused multiply-add is disabled. GPS-time bit-exactness across arbitrary tool pipelines is not guaranteed; drift is indistinguishable from tampering by design, with profile scoping as the mitigation. We measured the one toolchain used here rather than assuming it: writing and re-reading 200 000 real Autzen points through LAS and through LAZ preserved every gpsTime float64 bit-identically (0 points differing, 0 ULP; `eval/results/gpstime-stability.json`). That is evidence for these containers at

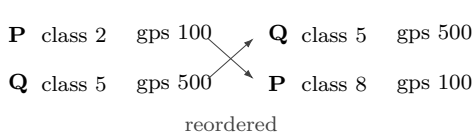
P and Q share a quantized cell and tie on intensity; only P.class is edited.

P = (class 2, gpsTime 100) Q = (class 5, gpsTime 500) edit: P.class 2 → 8

(a) Shared canonical order

Every attribute root inherits the same serialized sequence.

Order key (Morton, intensity, class, gpsTime, . . .)



Root	Outcome	Interpretation
classification	**changes**	intended edit
gpsTime	**changes**	contamination
lower-priority differing attributes	**may change**	sequence spillover

Unrelated-root contamination **500 / 500**

(b) Attribute-specific projections

Each root is sorted by Morton plus its own attribute.

Root / sort key	Before	After	Outcome
classification (Morton, class)	P:2 Q:5	**Q:5** **P:8**	**changed**
gpsTime (Morton, gpsTime)	P:100 Q:500	P:100 Q:500	unchanged
record (Morton, full tuple)	P:(2,100) Q:(5,500)	Q:(5,500) P:(8,100)	**changed**

Unrelated-root contamination **0 / 500**

Edit detection **500 / 500**

Fig. 3 When v1's per-attribute isolation holds, and when it does not. Contamination requires two points in one quantized cell that also tie on every sort key of higher priority than the edited attribute; a classification edit can then reorder them and move unrelated roots. Constructed to force that tie, 500/500; at realistic coincident-cell densities, 1/3,600. Profile v2's per-attribute projection roots remove the dependence on a single order, so only the edited and record roots move (0/500), and the edit is still detected (500/500)

these tool versions, not a general guarantee: a producer that re-quantizes or narrows gpsTime changes the canonical bytes and therefore the identity, which the profile treats as a derived-from transform. Reprojection is out of identity scope precisely because reprojection libraries are not bit-reproducible. All localization and identity claims reduce to SHA-256 and the domain separation of Sect. 4; we add no novel cryptographic assumption.

What this supports in practice. Two consequences follow from a deterministic, profile-scoped identity, and neither needs anything beyond what is evaluated here. The first is recognition: a custodian holding several files computes each one's identity under a single declared profile and learns whether they are the same survey — the everyday question a file hash cannot answer. The second is change localization across versions: when two copies differ, the per-attribute roots say which spatial chunks and which

attribute domains moved, so a reviewer inspects the part that changed instead of re-validating the whole cloud. Both are statements about integrity — what the canonical content is, and where it differs — and neither interprets the difference. The scheme does not judge whether a re-classification was correct; it reports that the classification channel changed, and where. Long-term archival integrity in public repositories is a natural application of the same property, since one citable identity survives the format migrations that invalidate a stored checksum.

8 Limitations and Future Work

The scheme does not detect sub-grid-step perturbation, a forged scene signed by a legitimate key, or key compromise; the latter two are inherited from any signature-based provenance system. It requires decode access to point data. Deferred, each with its trigger: external timestamping (contested signing time), a registry or transparency log of manifests (cross-organization discovery; RFC 9162 is the template), and a publicly citable CI run URL (the workflow already passes on GitHub's native Linux runner). A content identity that is stable across encodings and localizes change per region and channel is also the kind of primitive that dataset synchronization and version tracking would need; we have not built either, and make no claim about them here.

Profile families beyond the geospatial grid. The identity model is one construction instantiated by a declared profile, and it extends to capture domains whose formats carry no integer grid, provided each extension keeps identity a verified per-profile property rather than a cross-format promise. A layered program adds, in order: float-native formats (PLY, PCD, PTX/PTS) under a mobile/local-capture profile, where identity is taken over coordinates *quantized onto the grid* and a numeric representation is identity-compatible only when every decoded coordinate maps unambiguously to the declared grid, otherwise the source yields a new identity linked through derived-from provenance; E57 and terrestrial scanning under a profile that records scan pose and registration as signed provenance, excluded from the base point-content identity, and defines identity per scan in its declared frame, so applying a pose yields a new identity unless a future profile defines an exact deterministic transformation procedure; and time-series LiDAR (ROS PointCloud2, MCAP, rosbag2) with a per-sweep Frame-ID and a Sequence Content-ID that Merkle-binds an ordered list of Frame-IDs, giving logarithmic "this frame, unaltered, at this ordinal position in the sequence" proofs over a drive. Meshes and structured scenes (OBJ, STL, USD, USDZ) are out of scope. Each profile is a normative module with its own vectors, malformed-input and non-equivalence tests, and conformance suite.

9 Conclusion

Under a declared canonicalization profile, a geospatial point cloud can carry a deterministic identity that survives content-preserving re-encoding and localizes changes that alter an in-scope domain's canonical representation to an attribute and a region, with authentication paths logarithmic in the number of spatial chunks. The identity is fragile precisely where watermarking is robust, needs no embedded signal, no service,

and no network, and its manifest rides existing provenance rails. The boundary of the claim — identity only under a profile, provenance never equal to truth — is stated in the abstract and enforced by the construction.

Statements and Declarations

Funding. This work was self-funded. No external funding or institutional support was received.
Competing interests. The author declares no competing interests.
Ethics approval. Not applicable. The study involves no human participants, animals, or personal data.
Data availability. The controlled evaluation uses author-generated synthetic clouds produced by a seeded generator that is released with the artifact under CC-BY-4.0, so those clouds regenerate byte-for-byte from the committed seeds. The real-data experiments use the public "Autzen Stadium" point cloud (`autzen-classified.copc.laz`), distributed by Hobu, Inc. through the PDAL project (https://github.com/PDAL/data/tree/main/autzen) under a CC-BY-4.0 license; the LiDAR was collected by Watershed Sciences, Inc. (2010) and classified by M. Sampson, Hobu, Inc. (2021). We reference this dataset by URL rather than redistributing it; the fetch script records the URL, access date, and SHA-256 checksum so the exact bytes can be retrieved.
Code availability. The profile specifications, reference implementation and its test suite, committed test vectors, evaluation harness and recorded results, continuous-integration workflow, and browser demonstrator are provided as **Online Resource 1** (`ESM_1.zip`).
Author contributions. A. Urias is the sole author and conducted all aspects of the work. ORCID: 0009-0007-3147-323X.
Use of generative AI. The author used a large language model to assist with drafting the manuscript and implementing the reference code. All results were reproduced and verified by the author, who reviewed the text and is responsible for its content.

References

ASPRS (2019) LAS Specification, Version 1.4-R15. American Society for Photogrammetry and Remote Sensing

Benet J (2014) IPFS: Content addressed, versioned, p2p file system. arXiv preprint https://arxiv.org/abs/1407.3561

Chen YC, Chou YP, Chou YC (2019) An image authentication scheme using merkle tree mechanisms. Future Internet 11(7):149. https://doi.org/10.3390/fi11070149

Clarke D, Devadas S, van Dijk M, et al (2003) Incremental multiset hash functions and their application to memory integrity checking. In: Advances in Cryptology — ASIACRYPT 2003

Coalition for Content Provenance and Authenticity (2026) C2PA Technical Specification, Version 2.4. 2.x line; version 2.1 tested. https://spec.c2pa.org/specifications/specifications/2.4/specs/C2PA_Specification.html

Destefanis T, Guliyeva S, Boccardo P, et al (2025) Advancing flood detection and mapping: A review of earth observation services, 3D data integration, and AI-based techniques. Remote Sensing 17(17):2943. https://doi.org/10.3390/rs17172943

Giaretta D (2011) Advanced Digital Preservation. Springer

Golaszewski E, Krawetz N, Sherman AT, et al (2026) Verifying provenance of digital media: Why the C2PA specifications fall short. arXiv preprint https://arxiv.org/abs/2604.24890

Hobu, Inc. (2021) Cloud Optimized Point Cloud Specification 1.0

Laurie B, Messeri E, Stradling R (2021) Certificate transparency version 2.0. Tech. Rep. RFC 9162, IETF, https://doi.org/10.17487/RFC9162, https://www.rfc-editor.org/rfc/rfc9162

Maitin-Shepard J, Tibouchi M, Aranha DF (2017) Elliptic curve multiset hash. The Computer Journal 60(4):476–490. https://doi.org/10.1093/comjnl/bxw053

Merkle RC (1988) A digital signature based on a conventional encryption function. In: Advances in Cryptology — CRYPTO '87, LNCS, vol 293. Springer, pp 369–378

Nemecek A, He H, Cheng G, et al (2026) Authenticated contradictions from desynchronized provenance and watermarking. In: Proc. IEEE/CVF CVPR Workshops (APAI), https://arxiv.org/abs/2603.02378

Open Geospatial Consortium (2026) LAZ 1.4 Community Standard (OGC 24-070r1)

Ozcelik I, Medury S, Broaddus J, et al (2021) An overview of cryptographic accumulators. In: Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP), pp 661–669, https://doi.org/10.5220/0010337806610669

rapidlasso GmbH (2025) LAZ Specification 1.4, Revision R1. Revision date 15 November 2025. https://downloads.rapidlasso.de/doc/LAZ_Specification_1.4_R1.pdf

Rundgren A, Jordan B, Erdtman S (2020) JSON canonicalization scheme (JCS). Tech. Rep. RFC 8785, IETF, https://doi.org/10.17487/RFC8785, https://www.rfc-editor.org/rfc/rfc8785

Schaad J (2022a) CBOR object signing and encryption (COSE): Initial algorithms. Tech. Rep. RFC 9053, IETF, https://doi.org/10.17487/RFC9053, https://www.rfc-editor.org/rfc/rfc9053

Schaad J (2022b) CBOR object signing and encryption (COSE): Structures and process. Tech. Rep. RFC 9052, IETF, https://doi.org/10.17487/RFC9052, https://www.rfc-editor.org/rfc/rfc9052

Watershed Sciences, Inc., Sampson M (2021) Autzen stadium sample point cloud (autzen-classified.copc.laz). Distributed by Hobu, Inc. via the PDAL project under CC-BY-4.0, https://github.com/PDAL/data/tree/main/autzen. LiDAR collected 2010; classification 2021

Xu R, Chen Y, Chen G, et al (2022) SAUSA: Securing access, usage, and storage of 3D point cloud data by a blockchain-based authentication network. Future Internet 14:354. https://doi.org/10.3390/fi14120354

Yang J, Lu X, Chen W (2022) A robust scheme for copy detection of 3D object point clouds. Neurocomputing 510:181–192. https://doi.org/10.1016/j.neucom.2022.09.008

Zaman KAU, Alam MZ, Ali MNM, et al (2025) Deep neural watermarking for robust copyright protection in 3D point clouds. Annals of Emerging Technologies in Computing 9(4):17–30. https://doi.org/10.33166/AETiC.2025.04.002

Zhang Z, Zhang L, Wang P, et al (2024) Robust watermarking algorithm based on mahalanobis distance and ISS feature point for 3D point cloud data. Earth Science Informatics 17(1):783–796. https://doi.org/10.1007/s12145-023-01206-1

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- ESM1.zip