BEGINNING OF TRANSMISSION

State: published
Pairing: complete (DOI, GitHub, SHA-256, Zenodo URL)

Release metadata

Field	Value
Title	ENTO: an ENcrypted, Typed, Omnitrack container format for multimodal research data
Version	0.4
Concept DOI	10.5281/zenodo.20396329
GitHub	https://github.com/docxology/entofile/releases/tag/v0.4
Zenodo	https://zenodo.org/records/20396329
SHA-256	`05bcfb08323ac820…`
SHA-512	pending

How to verify

- Scan **Integrity** QR and compare the embedded SHA-256 prefix to the table above.
- Scan **Zenodo / GitHub** QR codes and confirm they resolve to this release pairing.
- Full hashes and structured fields: `../data/transmission_manifest.json`.

Metadata
Citation
Contact
Integrity
Code128
Zenodo
GitHub
Manifest
T:ENTO: an ENcrypted, Typed, Omnitra

Figure 1: Integrity QR strip

Structured manifest: `../data/transmission_manifest.json`

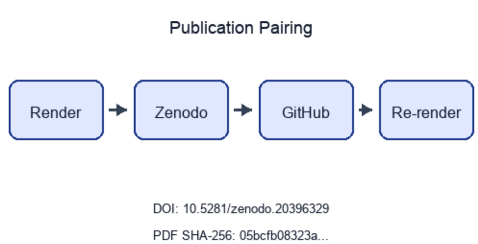

Figure 2: Publication pairing flow

ENTO: an ENcrypted, Typed, Omnitrack container format for multimodal research data

Authenticated ZIP containers for observability-graded research data

Daniel Ari Friedman
Active Inference Institute
FractAI
daniel@activeinference.institute
ORCID: 0000-0001-6232-9096
DOI: 10.5281/zenodo.20396329

June 11, 2026

Contents

1 Abstract

ENTO — an **EN**crypted, **T**yped, **O**mnitrack container — is a flat ZIP archive [zip, 2024] for bundling heterogeneous research artifacts (time series, genomics slices, spectrograms, provenance proofs) into one verifiable file. It combines typed *omnitrack* manifests, per-track authenticated encryption with AES-256-GCM on format 0.4.0 (suite aes-256-gcm) grounded in AES, AEAD, and GCM standards and security analysis [National Institute of Standards and Technology, 2001, McGrew, 2008, Dworkin, 2007, McGrew and Viega, 2004], 4 graded observability levels that control how much manifest metadata a recipient sees, and an optional minimal hash-chained proof export aligned with PROV-style derivation and tamper-evident log lineage [Merkle, 1988, Haber and Stornetta, 1991, W3C, 2013, Lebo et al., 2013]. The reference implementation lives under `src/` with benchmarks orchestrated by `scripts/ento_analysis.py`; release facts and measured values are injected from pipeline outputs rather than hand-edited prose, following reproducible-computing practice around scripted analyses, versioned inputs, and accessible run artifacts [Sandve et al., 2013, Wilson et al., 2017].

Release 0.4 is the manuscript release candidate for default ENTO wire format 0.4.0. The default writer emits 0.4.0; the prior supported formats (0.2.0, 0.3.0 and 0.3.1) remain explicit compatibility choices rather than disappearing from the decrypt path.

Compared with array stores (HDF5 [hdf, 2024], Zarr [zar, 2023]) and publication containers (EPUB [W3C, 2023]), ENTO emphasizes authenticated track envelopes and graded manifest export rather than chunk-level partial I/O or rendering semantics. Preservation frameworks such as OAIS and PREMIS define archive responsibilities and preservation metadata [Consultative Committee for Space Data Systems, 2024, PREMIS Editorial Committee, 2015], while RO-Crate [roc, 2024, Soiland-Reyes et al., 2022] and BagIt [Kunze et al., 2018] provide complementary research-object and transfer packaging. ENTO occupies the narrower file-format layer: it specifies ciphertext layout in `data/ento_track_header.ksy` (nonce(12) || tag(16) || ciphertext) using a Kaitai Struct definition [Kaitai Struct, 2026] and validates manifests with JSON Schema [jso, 2020]. Verification deliberately separates key-authenticated integrity from keyless corruption detection, and 3 compatibility formats remain version-dispatched alongside the default. The default profile includes associated-data binding [McGrew, 2008] and PADMÉ length padding [Nikitin et al., 2019], bounding the on-disk ciphertext member-size channel to a coarse bucket. Length is never hidden exactly: at non-sealed observability levels the manifest's cleartext `byte_length` discloses it directly, and even under sealed export the padded member size still reveals the PADMÉ bucket.

This configuration runs 150 repetitions across observability levels 0,1,2,3. The benchmark suite reports 2400 rows with mean pack throughput 78.9296 MiB/s on the medium-track condition at observability level 3 ($n = 150$, CV 15.3%; wall-clock throughput is re-measured each run, so we report its dispersion and make no superiority claim), mean unpack latency 0.001130 s, and an exact, zero-variance ciphertext expansion of 1.7113 on fixture tracks that follows $r(n) = (H + \mathrm{PADME}(n + 8)) / n$. Tamper detection succeeded on 2400 benchmark rows (rate 1.0; validation status pass). Manifest sizes for the EEG fixture range from 311 bytes (sealed) to 499 bytes (auditable).

Keywords: authenticated encryption, research containers, reproducible pipelines, multimodal data, observability, FAIR-adjacent packaging

2 Introduction

Multimodal research workflows produce heterogeneous artifacts: time series, genomics slices, spectrograms, and provenance proofs. General-purpose archives (ZIP [zip, 2024], HDF5 [hdf, 2024], Zarr [zar, 2023]) and document containers (EPUB [W3C, 2023], Matroska [mat, 2024]) each cover part of the problem–layout, array semantics, or publication packaging–but none combine typed track ontologies, per-track authenticated encryption, version-dispatched on-disk formats, graded observability, and an offline reference CLI in one flat ZIP envelope. Preservation standards define a wider institutional setting: OAIS frames archival responsibilities and information packages [Consultative Committee for Space Data Systems, 2024], and PREMIS records preservation objects, events, rights, and agents [PREMIS Editorial Committee, 2015]. ENTO does not replace those systems. It targets the lower file-format layer that they could store, transfer, describe, or cite. Research-object packaging standards such as RO-Crate [roc, 2024, Soiland-Reyes et al., 2022] and BagIt [Kunze et al., 2018] emphasize FAIR metadata [Wilkinson et al., 2016] and transfer checksums without mandating a fixed ciphertext header per track.
Policy-bound formats including OpenTDF [ope, 2024] demonstrate that decryption can depend on external attributes; ENTO instead documents explicit observability levels so recipients know which manifest fields appear at export time. Scientific audit trails often cite PROV-DM or PROV-O [W3C, 2013, Lebo et al., 2013] and CADF [cad, 2013]; ENTO optional proof export supplies a minimal hash chain over track digests, drawing on hash-linked authentication and timestamping lineage [Merkle, 1988, Haber and Stornetta, 1991], without prescribing a ledger.
The name ENTO is an acronym for the format's three defining properties: it is **EN**crypted (every track is sealed under an authenticated cipher), **T**yped (every track declares an ontology URI that downstream tools can interpret without decrypting), and **O**mnitrack (one flat archive carries arbitrarily many heterogeneous tracks side by side). This work targets the gap between preservation envelopes and encrypted research payloads. Concretely, an ENTO file is a ZIP archive holding a single `manifest.json`, one encrypted `tracks/{id}.ento` member per track, and an optional `proof/chain.json`. Tracks carry URIs such as `ento:timeseries.eeg` and `ento:genomics.vcf`; manifests validate against `data/ento_manifest_schema.json` using JSON Schema [jso, 2020]; ciphertext uses a fixed binary header documented in `data/ento_track_header.ksy`. The project follows the template code-centric layout: algorithms in `src/`, thin orchestrators in `scripts/`, and manuscript variables injected from `output/data/ento_benchmark_results.csv`.

2.1 Design goals

1. **Interoperable envelope** — any ZIP tool can list `manifest.json` and `tracks/*.ento` entries [zip, 2024].
2. **Typed multimodal tracks** — ontology URIs with optional resolution metadata (sec. 4).
3. **Authenticated confidentiality** — AES-256-GCM on default format 0.4.0 with suite aes-256-gcm, associated-data binding as defined by the AEAD interface, and explicit compatibility dispatch for 0.2.0, 0.3.0 and 0.3.1 [National Institute of Standards and Technology, 2001, McGrew, 2008] (sec. 3).
4. **Graded export** — observability levels redact manifests without re-encrypting payloads (sec. 5).
5. **Reproducible evidence** — scripted benchmarks, claim ledger, and registry-backed manuscript tokens, aligned with reproducible-computational-research guidance that scripts, runs, data classes, and results remain inspectable [Sandve et al., 2013, Wilson et al., 2017] (sec. 12).

2.2 Reader's guide

- sec. 3 defines the cryptographic and manifest contract.
- sec. 7 states the container model, the ciphertext-expansion law, and the integrity and observability predicates formally.
- sec. 6 documents verify-before-unpack and pipeline verification gates.
- sec. 4 lists ontology URIs and fixture digests.
- sec. 5 documents observability levels and proof export.
- sec. 8 reports benchmark figures generated by the analysis pipeline.
- sec. 9 interprets throughput, expansion, and tamper metrics.
- sec. 11 lists fixture tracks and CLI entry points.
- sec. 12 records configuration hash and artifact inventory.
- sec. 14 states threat model and non-goals.

3 Methodology

The benchmark matrix, figure-filter contract, and visualization pipeline are documented in `docs/methods.md`.
The method starts from four hard constraints. First, an ENTO file is still ZIP: member names, sizes, and central-directory metadata are inspectable before any key is supplied. Second, AES-GCM is an AEAD primitive, so adversarial integrity comes from successful keyed authentication, not from unkeyed JSON digests [McGrew, 2008, Dworkin, 2007, McGrew and Viega, 2004]. Third, wall-clock benchmark columns depend on host load and must be measured with dispersion rather than turned into reproducibility anchors. Fourth, the paper release label and wire-format string are distinct: paper 0.4 documents default wire format 0.4.0, while 0.2.0, 0.3.0 and 0.3.1 remain compatibility formats.

3.1 Container layout

An ENTO file is an ordinary ZIP archive with a small, fixed set of members, so any ZIP tool can enumerate its contents even without the decryption key:

```
container.ento.zip
  manifest.json         # typed index: format_version, observability_level, per-track metadata
  tracks/
     eeg.ento          # one authenticated-encrypted member per track
     vcf.ento
     spectrogram.ento
  proof/
     chain.json         # optional hash chain (omitted at the sealed level)
```

`manifest.json` is the typed index a recipient reads first; each `tracks/{track_id}.ento` member is a self-describing authenticated-ciphertext blob; `proof/chain.json` is an optional integrity chain. The directory layout is deliberately flat — there is no central directory format beyond ZIP's own, so partial inspection degrades gracefully to "list the names."

3.1.1 Per-track binary header

Every `.ento` member is the concatenation `nonce || tag || ciphertext`:

```
byte 0                12                  28                          end
    AES-GCM nonce        AES-GCM auth tag      authenticated ciphertext


       12 bytes             16 bytes                variable length
```

The first 12 bytes are the AES-256-GCM nonce and the next 16 bytes are the authentication tag for `format_version` **0.4.0** (a fixed 28-byte header in total), followed by the ciphertext body. Under the default profile, that body is the PADMÉ-padded plaintext with an original-length prefix; compatibility formats without padding keep a length-preserving body. The canonical machine-readable definition is the Kaitai Struct spec in `data/ento_track_header.ksy`, using the `.ksy` declarative binary-format language [Kaitai Struct, 2026]. Decryption recomputes and verifies the tag *before* releasing any plaintext — a corrupted or forged byte makes the whole track fail closed (sec. 6). This layout fixes the version-aware ciphertext-expansion law r(n) = (H + PADME(n + 8)) / n (eq. 4 in sec. 7).

3.1.2 Worked example

To build a container, `pack` derives a per-track key, encrypts each track, writes the `tracks/*.ento` members and a full internal `manifest.json`, redacts the manifest to the requested observability level, and (above the sealed level) appends `proof/chain.json`:

```
uv run python scripts/ento_cli.py genkey -o master.key
uv run python scripts/ento_cli.py pack -k master.key -o study.ento.zip \
    --observability 3
uv run python scripts/ento_cli.py verify -i study.ento.zip -k master.key   # key-authenticated
uv run python scripts/ento_cli.py unpack -i study.ento.zip -k master.key -o ./out
```

The reference `pack` encrypts the project's typed fixture tracks (resolved through `src/ontology.py`; supply `--fixtures` to point at another set) and emits the ZIP container redacted to `--observability 3` (auditable). The recipient runs `verify` before `unpack`; with the master key present, verification authenticates every track through the GCM tag rather than trusting any unkeyed manifest field (sec. 6).

3.2 Cryptography

The reference implementation in `src/crypto.py` derives per-track keys with HKDF-SHA256 [Krawczyk and Eronen, 2010, National Institute of Standards and Technology, 2015] (`info = "ento:track:{track_id}"`). HKDF is used here as a key-separation tool: one master key enters the container workflow, but each track receives a distinct subkey labelled by its track id.

Master keys are 32 random bytes from `genkey`. The default writer emits format 0.4.0; the decrypt path is version-dispatched across 4 supported AES-256-GCM formats so default and compatibility containers remain readable. A fresh nonce is drawn per encryption under each per-track key because GCM treats the nonce like a one-time label for that key; reusing it can expose plaintext relationships and enable forgeries [Joux, 2006, Böck et al., 2016]. Nonce-misuse-resistant AEADs such as AES-GCM-SIV are a relevant future design alternative for deployments that cannot bound nonce uniqueness operationally, but they are not implemented in ENTO 0.4.0 [Gueron and Lindell, 2019].

`format_version`	Encryption	Library
0.4.0 (default)	AES-256-GCM with associated-data binding and PADMÉ padding [National Institute of Standards and Technology, 2001, McGrew, 2008, Dworkin, 2007, McGrew and Viega, 2004, Nikitin et al., 2019]	`cryptography` (`src/crypto_gcm.py`)
0.2.0, 0.3.0 and 0.3.1 (compatibility)	Version-dispatched AES-256-GCM profiles, including no-AAD, AAD-bound, and PADMÉ-padded variants	`cryptography` (`src/crypto_gcm.py`)

AEAD means the ciphertext and selected cleartext context are checked together. For 0.4.0, the associated data is a small label containing the format version and track id; it is not encrypted, but changing it causes the GCM tag check to fail [McGrew, 2008, Dworkin, 2007]. Decryption authenticates that tag before releasing plaintext (fail closed) [Ferguson et al., 2010]. Unpack and `verify` also compare SHA-256 plaintext and ciphertext digests when digest fields are present [National Institute of Standards and Technology, 2015], but those digests are unkeyed corruption checks rather than substitutes for AEAD authentication.

3.3 Container verification

The CLI exposes `verify -i container.ento.zip` for keyless integrity checks (schema, ZIP member set, ciphertext digests, proof binding). Supply `-k master.key` to confirm plaintext digests without writing output files. See sec. 6.

3.4 Verification vectors

Pinned regression vectors in `data/test_vectors/` lock HKDF and GCM backends across refactors (`tests/test_crypto_vectors.py`, `tests/test_crypto_gcm.py`).

3.5 Manifest schema

`manifest.json` validates against Draft-07 JSON Schema in `data/ento_manifest_schema.json`. The schema accepts the supported format set (0.2.0, 0.3.0, 0.3.1, 0.4.0); the default writer uses **0.4.0** unless a compatibility format is selected explicitly. Required fields include `observability_level` and per-track `type`, `sha256_plaintext`, `sha256_ciphertext`, and `byte_length`. Proof export hashes the exact JSON bytes emitted by `manifest_to_json`; the JCS specification defines a general JSON canonicalization scheme, but ENTO does not claim JCS interoperability in this release [Rundgren et al., 2020].

3.6 Manifest footprint across tracks

fig. 3 shows how exported manifest size changes with observability level for each committed fixture track (`eeg`, `vcf`, `spectrogram`) under `small_tracks_r0`. The plot makes the redaction trade-off visible across modalities without re-encrypting track payloads.

fig. 4 makes the field-level policy explicit: observability changes exported manifest fields, not the encrypted track members. It is therefore a metadata control layered beside, not instead of, AEAD verification.

3.7 Ciphertext overhead

fig. 5 decomposes per-track ciphertext bytes into the fixed 28-byte AEAD header and the remaining ciphertext body for `small_tracks_r0` at observability level 3, aligning with the track layout in sec. 3.

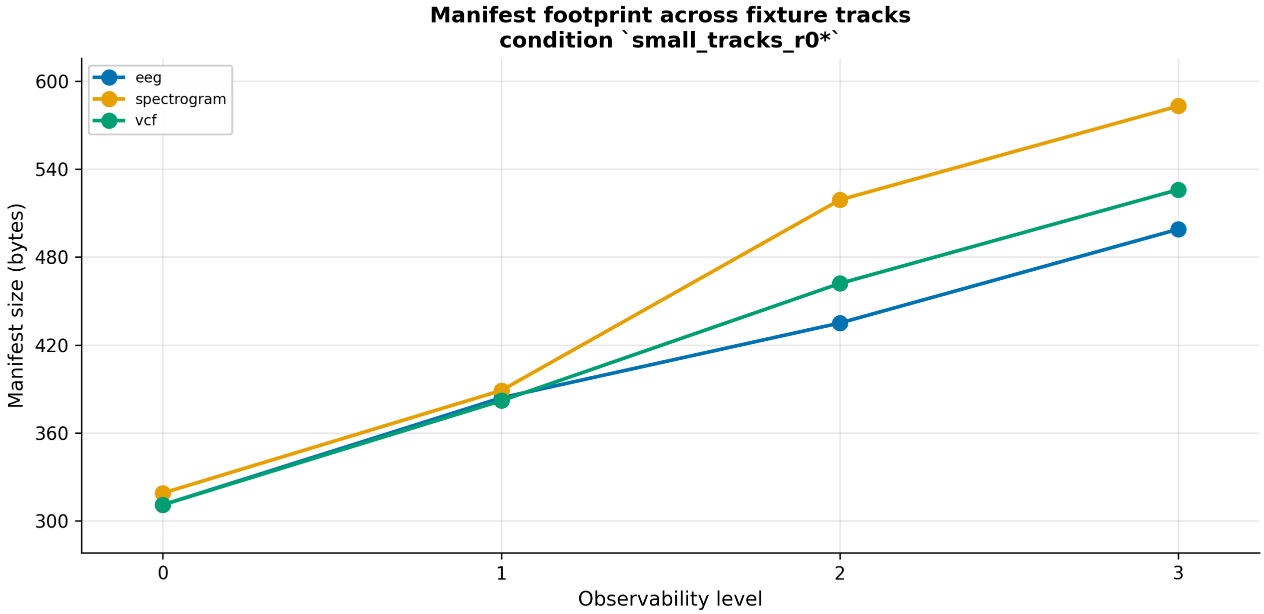

Figure 3: Exported manifest size, in bytes, against observability level for all three fixture tracks (eeg, vcf, spectrogram) on one axis, one line per track. The curves shrink in parallel as the export level drops, showing that graded redaction behaves uniformly across heterogeneous modalities rather than favouring any one track type. Filters: condition `small_tracks_r0*`. Data from `ento_benchmark_results.csv`. Generated by `generate_manifest_multitrack_figure` in `src/figures.py`.

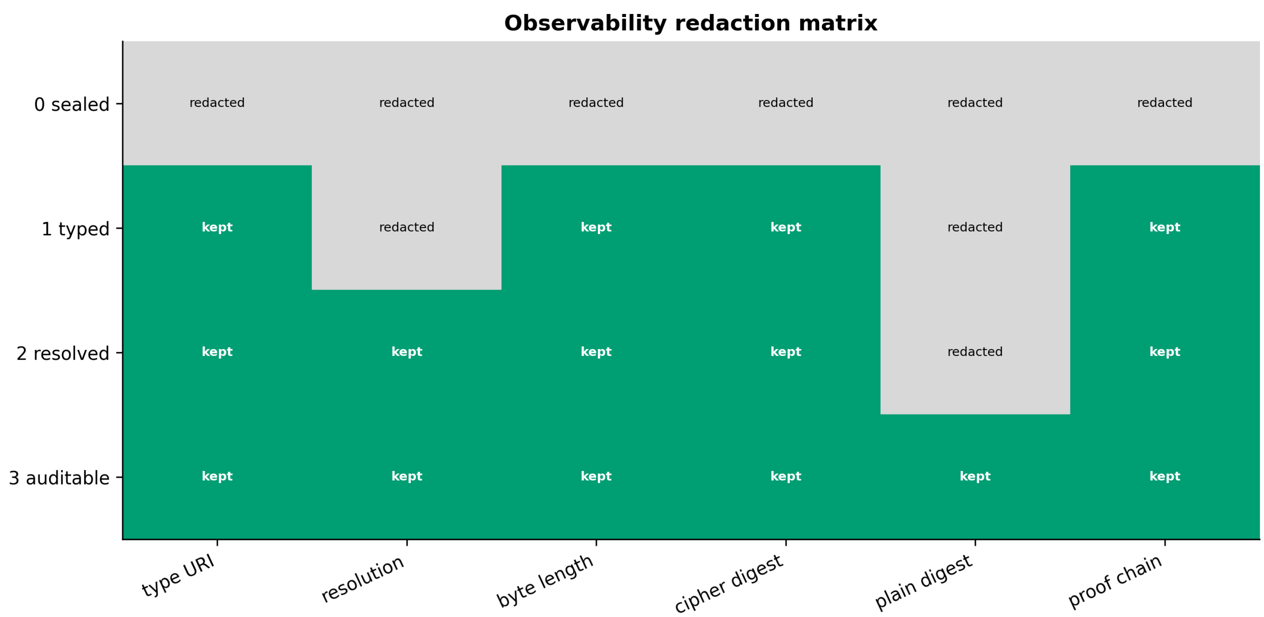

Figure 4: Manifest field-presence matrix for observability levels 0 through 3. The figure separates metadata redaction from cryptographic protection: lower levels remove type, resolution, digest, and declared length fields, while payload confidentiality and integrity are enforced by the encrypted track member. Filters: all benchmark rows. Data from `ento_benchmark_results.csv`. Generated by `generate_observability_redaction_matrix_figure` in `src/figures.py`.

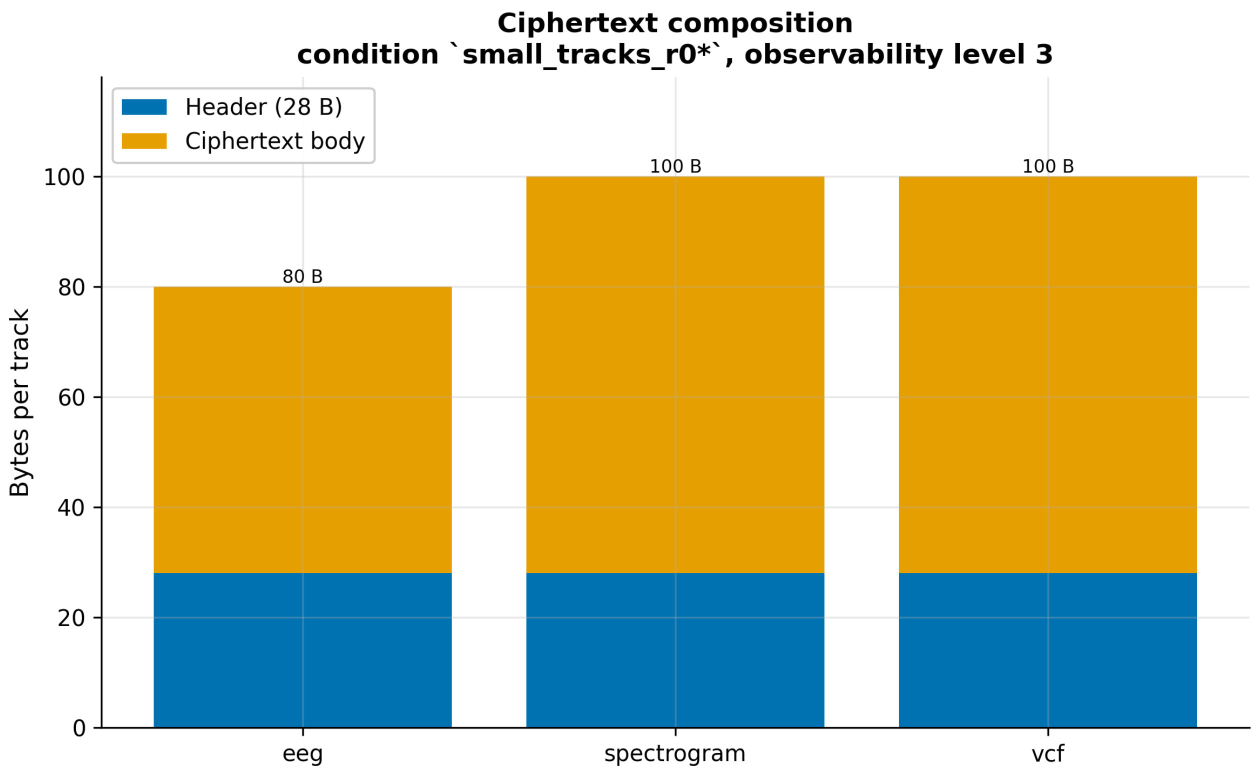

Figure 5: Per-track ciphertext decomposed into its two parts: the fixed 28-byte default 0.4.0 AEAD header (nonce plus authentication tag, bottom segment) and the variable ciphertext body (top segment), stacked per fixture track. The body includes PADME length padding under the default profile, so the bar shows both fixed authentication overhead and bucketed length-hiding overhead. Filters: condition `small_tracks_r0*`, observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_crypto_overhead_figure` in `src/figures.py`.

3.8 Observability levels

Level	Name	Exported manifest content
0	sealed	Track ids and byte lengths only
1	typed	Adds track type URIs
2	resolved	Adds resolution descriptors
3	auditable	Full hashes and plaintext digests

Filtering is centralized in `src/observability.py`; pack always writes a full internal manifest before export redaction at the requested level (up to 3).

3.9 Proof export

`src/proof.py` emits a hash-chained JSON structure over track plaintext digests, in the same broad lineage as Merkle hash-linked authentication and Haber–Stornetta timestamping [Merkle, 1988, Haber and Stornetta, 1991]. `verify_proof_export` recomputes `manifest_sha256` over exported manifest bytes, walks the chain, and verifies the links correspond to the manifest's tracks (see sec. 5).

4 Ontology and fixtures

ENTO registers track types as stable URIs in `src/ontology.py`. Each URI maps to required resolution keys validated before pack, so manifests cannot claim a genomics track without the chromosome/build metadata the ontology expects.

4.1 URI registry

URI	Label	Required resolution keys
`ento:timeseries.eeg`	EEG time series	`hz`
`ento:genomics.vcf`	VCF genomics slice	`build`, `chr`
`ento:spectrogram`	Spectrogram matrix	`hz`, `n_fft`, `shape`
`ento:blockchain.proof`	Proof chain anchor	(none)

New types extend the registry without changing the ZIP layout: manifests reference the URI string; exporters may redact resolution fields at lower observability levels (sec. 3). The URI table is the contract between producers and downstream tools that interpret manifests without opening ciphertext.

4.2 Committed fixtures

The benchmark pipeline loads 3 deterministic tracks from `data/fixtures/` via `src/fixtures.py`:

File	Track id	SHA-256 (plaintext)
`eeg.csv`	`eeg`	`49a0f244623e895ec62a72579b8488cb` `448bc97a5294fa2d4767224d9ab385d5`
`sample.vcf`	`vcf`	`e78a6df9053dafc26e73f5ae5e521f68` `714bea7433e9aa7ab8df1e6219595e02`
`spectrogram.bin`	`spectrogram`	`fdeab9acf3710362bd2658cdc9a29e8f` `9c757fcf9811603a8c447cd1d9151108`

Missing fixtures fail closed when `require_all=True` (CLI pack and benchmark entry points). The spectrogram fixture is a small square deterministic byte matrix documented in `data/fixtures/README.md`. Fixtures anchor expansion and manifest-size figures (sec. 8) to bytes that remain in version control.

Fixture digests bind manuscript claims to committed bytes; regenerating fixtures updates both benchmarks and the `FIXTURE_EEG_SHA256`, `FIXTURE_VCF_SHA256`, and `FIXTURE_SPECTROGRAM_SHA256` tokens in `output/data/manuscript_variables.json`.

5 Proof export and observability

Two mechanisms govern what a recipient sees without re-encrypting tracks: **observability levels** filter exported manifests; **proof export** optionally emits a hash chain over track digests.

5.1 Observability redaction

Internal pack always writes a full auditable manifest. Export applies `src/observability.py::filter_manifest` at the requested level (0 through 3):

Level	Name	Visible fields
0	sealed	Track ids, byte lengths
1	typed	+ type URIs
2	resolved	+ resolution descriptors
3	auditable	+ SHA-256 digests

Benchmark manifest sizes for the EEG fixture (bytes, averaged across repetitions):

Level	Manifest bytes
0	311
1	384
2	435
3	499

fig. 16 plots the same sweep. CLI `--observability` controls export redaction only; stored ciphertext is unchanged. Level 0 is appropriate when filenames alone would leak too much context; level 3 supports reproducibility checks against fixture digests in sec. 4.

5.2 Proof chain

`src/proof.py` builds `proof/chain.json` with a deliberately minimal, unsigned hash-chain construction. It is conceptually adjacent to Merkle hash-linked authentication and Haber–Stornetta timestamping [Merkle, 1988, Haber and Stornetta, 1991], but it is not a ledger or a signature scheme:

1. `manifest_sha256` over the exact exported manifest JSON bytes
2. Per-track links hashing `(track_id, sha256_plaintext, previous_hash)` with SHA-256 [National Institute of Standards and Technology, 2015]

Hash-over-JSON schemes need a stable byte representation. ENTO uses the exact bytes emitted by `manifest_to_json` (`sort_keys=True`, fixed indentation, trailing newline) as the proof binding. The JCS specification is the relevant canonicalization standard for broader JSON interoperability, but ENTO does not implement or claim JCS compliance in this release [Rundgren et al., 2020].

`verify_proof_export(proof, manifest_json)` performs three checks—it recomputes the manifest digest, walks the hash chain, and confirms the links correspond one-to-one (by `track_id` and `sha256_plaintext`, in order) to the manifest's tracks with a matching `format_version`. Tampering with `manifest.json` after export fails the digest binding; a chain whose links describe a different track set than the manifest fails the correspondence check even when the individual link hashes are internally consistent.

The CLI exposes `verify -i container.ento.zip` for keyless checks (schema, ZIP member set, ciphertext digests, proof when present). Supply `-k master.key` to confirm plaintext digests without writing output files. Unpack repeats the same gate before decryption.

At observability level **sealed** (0), proof export is omitted: there is insufficient public metadata to anchor without revealing types or digests.

5.3 PROV alignment

Proof links are compatible with W3C PROV entity-activity patterns and PROV-O vocabulary alignment [W3C, 2013, Lebo et al., 2013]: each link records a derivation step over track content digests without mandating a particular ledger implementation.

6 Security verification

ENTO separates **integrity verification** from **decryption**. Operators and CI should treat third-party `.ento.zip` archives as hostile until `verify` succeeds.

The verification model deliberately avoids a common false comfort: a parseable ZIP and matching unkeyed digests are not the same as adversarial authenticity. An attacker who can rewrite the archive can also rewrite unkeyed hashes. A successful keyed AEAD check is different: the GCM tag can be recomputed only by a party with the correct per-track key and associated-data inputs. The repository can prove local parser discipline, keyed AEAD failure on tamper, and artifact/render consistency. It cannot, by itself, prove who built a public release, where the master key was stored, or whether an operator routed failures to a SOC; those are external controls named rather than implied.

6.1 Verify-before-unpack

The CLI subcommand `verify -i container.ento.zip` checks JSON Schema, ZIP member limits, exact member-set equality with the manifest, per-track ciphertext digests, and optional proof binding—without decrypting. Supply `-k master.key` to confirm plaintext SHA-256 fields when the export level includes them.

```
uv run python -m src.cli verify -i container.ento.zip
uv run python -m src.cli verify -i container.ento.zip -k master.key --require-proof
```

`unpack` repeats the same checks before writing plaintext. Failures emit a structured JSON audit line on stderr (event `ento.verify.failed`) for log aggregation.

6.2 Integrity assurance levels

`verify` reports exactly what it established, never more. Its `integrity` field takes one of 3 values:

- **key-authenticated** — every track decrypted under AES-256-GCM (master key supplied) and all plaintext digests matched. This is the only level that resists an adversary who controls the container bytes; it authenticates the track plaintext and the track identity (bound through per-track key derivation, and through AAD for AAD-bound formats), not the unkeyed manifest header fields.
- **digest-only** — no key supplied, but every ciphertext digest was present and matched. This detects *accidental* corruption only: the manifest digests and the hash-chained proof are unkeyed, so a motivated attacker who rewrites the archive can recompute them.
- **unverified** — no key, and at least one ciphertext digest was absent (a redacted or stripped manifest); nothing about the track bytes was checked.

`verify` fails closed by default—an `unverified` result exits non-zero unless `--allow-unverified` is given. The proof chain is a consistency structure binding a proof to a manifest, **not** an authentication of origin: adversarial integrity comes only from decrypting with the master key.

6.3 ZIP ingestion limits

Reference defaults in `src/security.py` cap archive size, member count, per-member **actual decompressed** bytes (bounded at read time rather than trusting the attacker-declared `file_size`), and an aggregate decompressed budget; duplicate member names are rejected before the membership check so a second blob cannot ride inside a duplicated allowed name. Oversize or malformed archives raise `ValueError` before crypto runs. Tests build real ZIP fixtures under `tmp_path` (no monkeypatch).

6.4 Tamper rejection outcomes

fig. 6 reports tamper-injection outcomes across the full benchmark matrix as a stacked share of detected versus missed rejections. Benchmark validation requires tamper detection rate 1.0 with status pass before release.

6.5 Container verification report

The analysis pipeline writes `output/reports/container_verification.json` with per-sample `verify_container` outcomes. `validate_generated_outputs()` requires all entries `ok: true` alongside 2400 tamper detections at rate 1.0 in `benchmark_validation.json`.

Latest gate sample: output/data/_bench_tmp/medium_3.ento.zip. Aggregate status: true.

6.6 Cryptography and the format ladder

All formats use AES-256-GCM (aes-256-gcm) via the audited `cryptography` library with HKDF-derived per-track keys. The default writer emits 0.4.0; 3 compatibility formats (0.2.0, 0.3.0 and 0.3.1) are version-dispatched and remain readable/writable via `pack --format`:

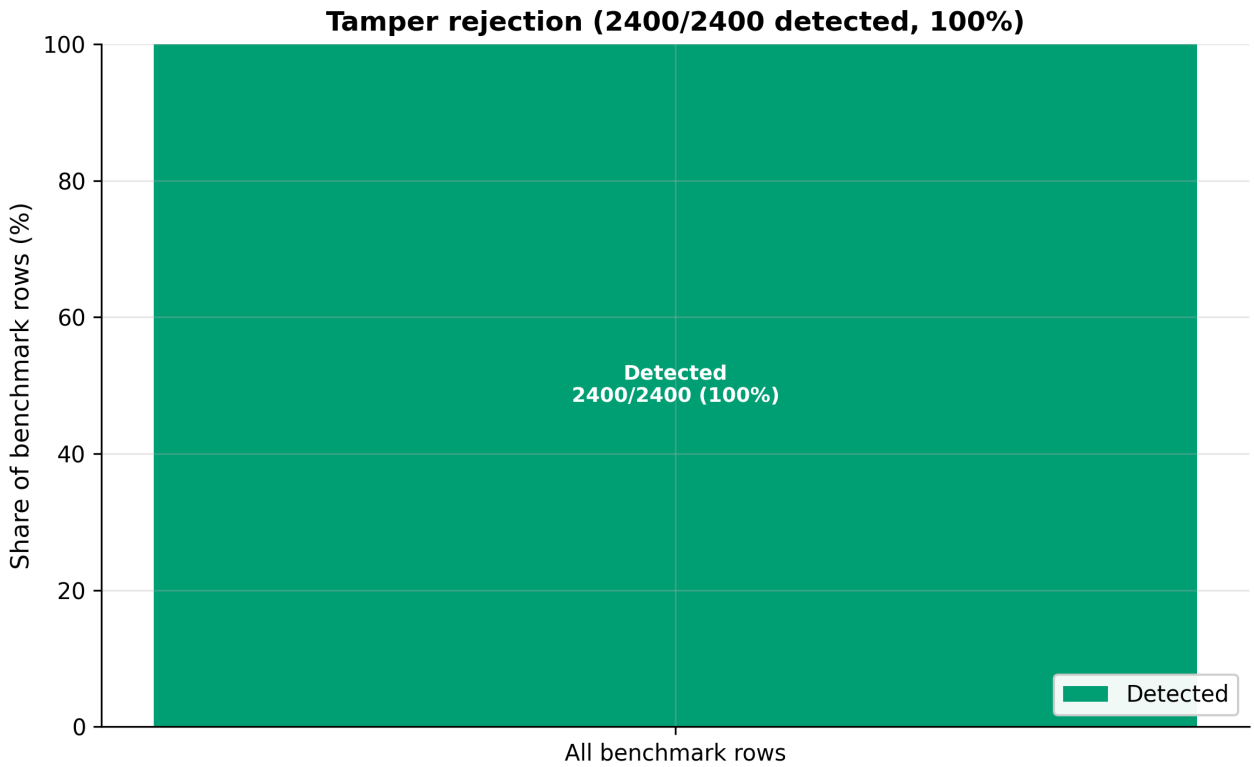

Figure 6: Tamper-injection outcomes across the full benchmark matrix as a 100%-stacked share of detected versus missed rejections. Each generated row flips one ciphertext tag byte and then attempts a key-based unpack, which must fail closed; the panel title reports the detected count and rate. A single solid 'detected' bar is the success condition for this generated matrix: keyed unpack rejected every injected tag-byte corruption. Filters: all benchmark rows. Counts use is_tamper_detected in src/benchmark_filters.py. Data from `ento_benchmark_results.csv`. Generated by `generate_tamper_figure` in `src/figures.py`.

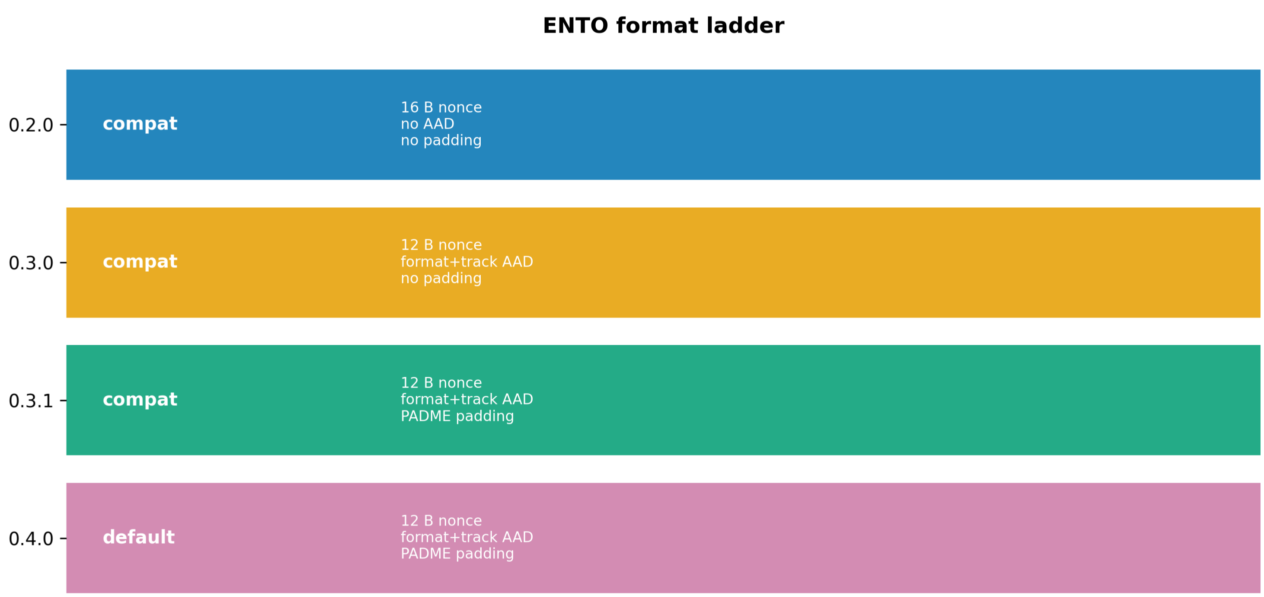

Figure 7: ENTO format ladder for supported AES-256-GCM wire formats. The default write format is 0.4.0; 0.2.0, 0.3.0, and 0.3.1 remain version-dispatched compatibility formats. Filters: all benchmark rows. Data from `ento_benchmark_results.csv`. Generated by `generate_format_ladder_figure` in `src/figures.py`.

format_version	Nonce	Associated data	Length-hiding
0.4.0 (default)	12-byte	binds format_version + track id	PADMÉ padding [Nikitin et al., 2019]
0.2.0, 0.3.0 and 0.3.1	version-dispatched	no-AAD through AAD-bound compatibility profiles	compatibility-dependent

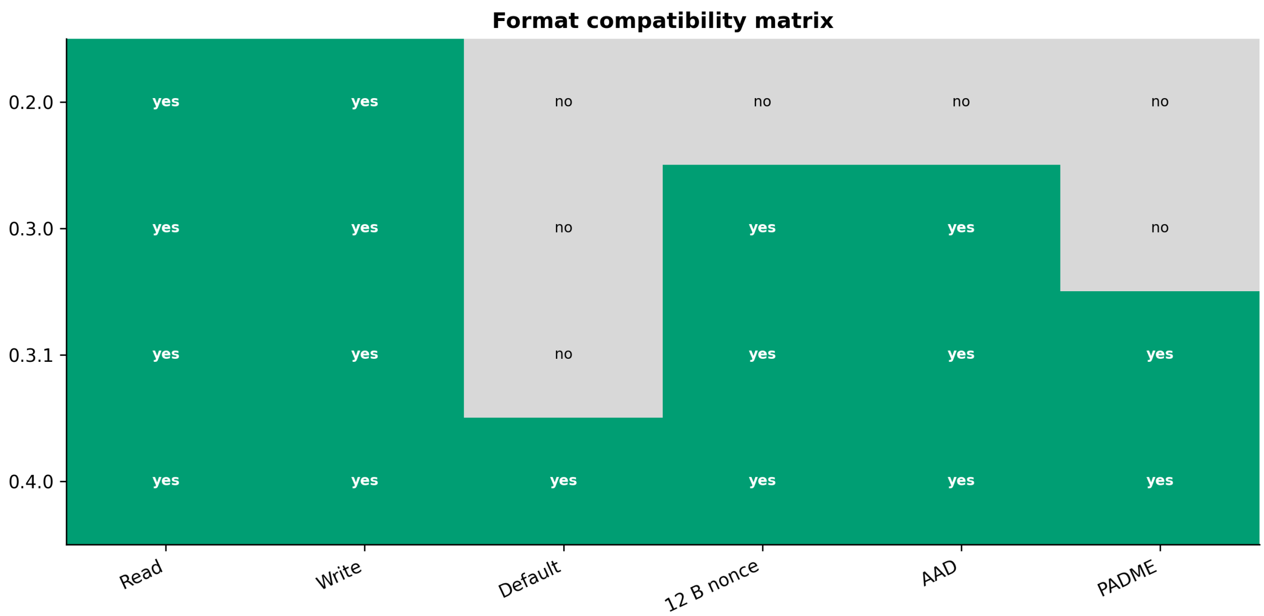

Figure 8: Compatibility matrix for every supported ENTO wire format. It distinguishes read/write support from the default writer choice and shows which formats carry the 12-byte nonce, associated-data binding, and PADME length padding. Filters: all benchmark rows. Data from ento_benchmark_results.csv. Generated by generate_format_compatibility_matrix_figure in src/figures.py.

fig. 7 and fig. 8 are the release-candidate guardrails: manuscript version 0.4 documents default wire format 0.4.0, and the compatibility formats remain explicit rather than implicit. Binding format_version in the associated data makes a format downgrade (including a padded↔unpadded swap) fail the GCM tag rather than mis-parse. fig. 9 bounds the length claim: default padding hides exact length only to PADMÉ buckets, while ZIP names and bucketed sizes remain visible. fig. 10 connects the supported-format claim to deterministic known-good and known-bad fixtures. Pinned vectors: data/test_vectors/hkdf_regression.json, aes256_gcm_regression.json. See sec. 3.

6.7 Nation-state deployment checklist

Production hardening beyond this reference implementation is documented in docs/nation_state_roadmap.md against NIST zero-trust, SSDF, key-management, and supply-chain-risk guidance, SLSA provenance levels, Sigstore signing, in-toto-style supply-chain attestations, CycloneDX SBOMs, MITRE ATT&CK detection mapping, and post-quantum standards work [Rose et al., 2020, NIST, 2022, National Institute of Standards and Technology, 2020, 2022, SLSA, 2024, Sigstore, 2026, Torres-Arias et al., 2019, OWASP CycloneDX, 2026, MITRE, 2026, National Institute of Standards and Technology, 2024b,a]. fig. 11 marks what this repository enforces (ZIP limits, verify-before-unpack, schema gates, deterministic artifact checks), what is partial (telemetry and release documentation), and what remains external/residual (artifact signing policy, HSM/KMS custody, SOC routing). Manuscript release 0.4 documents format **0.4.0** as the default on-disk contract, with 0.2.0, 0.3.0 and 0.3.1 retained as compatibility formats.

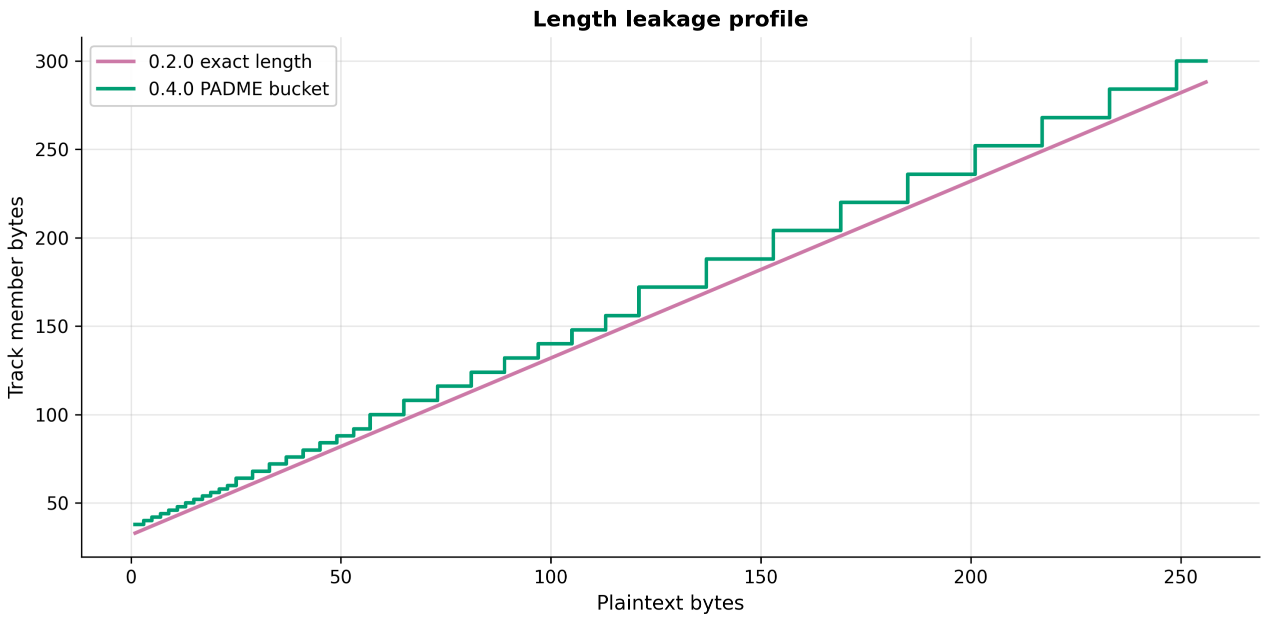

Figure 9: Track member bytes versus plaintext bytes for compatibility 0.2.0 and default 0.4.0. The 0.2.0 line rises one-for-one with plaintext length, while the 0.4.0 step profile reveals only PADME buckets plus the fixed AEAD header. The bucket size remains visible; this is mitigation of exact-length disclosure, not total traffic-analysis resistance. Filters: all benchmark rows. Data from `ento_benchmark_results.csv`. Generated by `generate_length_leakage_profile_figure` in `src/figures.py`.

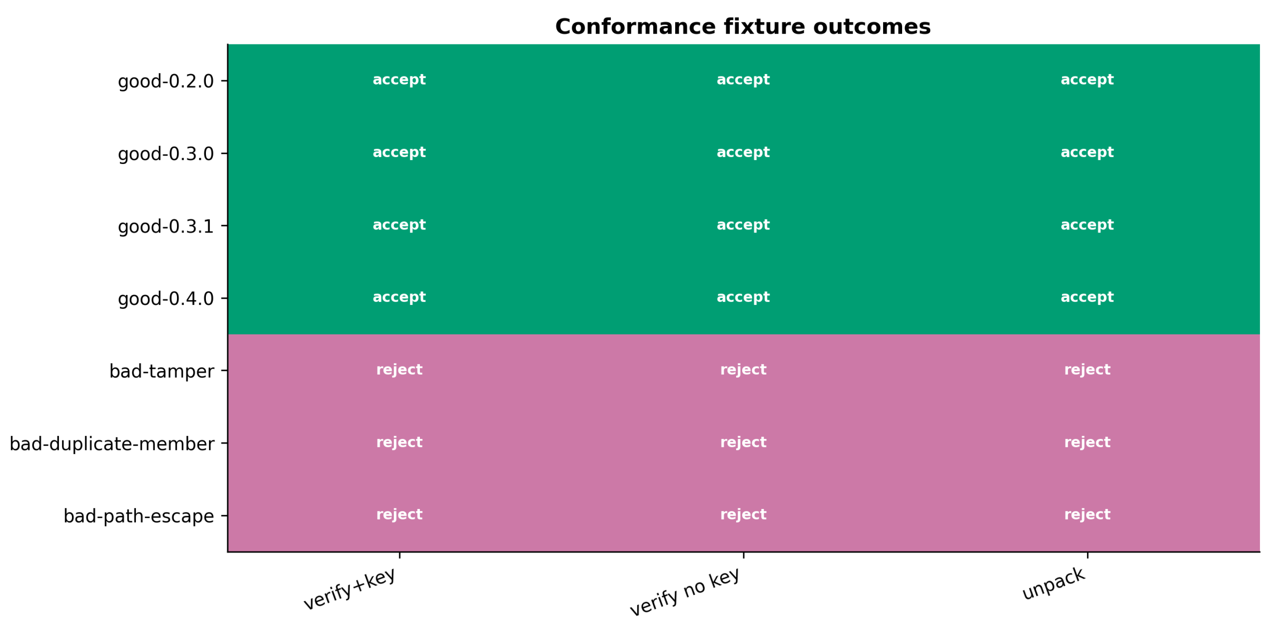

Figure 10: Conformance fixture matrix for known-good and known-bad ENTO containers. Valid containers for every supported format must verify and unpack; tamper, duplicate-member, and path-escape fixtures must fail closed. The cases are generated deterministically by `src/conformance.py`. Filters: all benchmark rows. Data from `ento_benchmark_results.csv`. Generated by `generate_conformance_outcomes_figure` in `src/figures.py`.

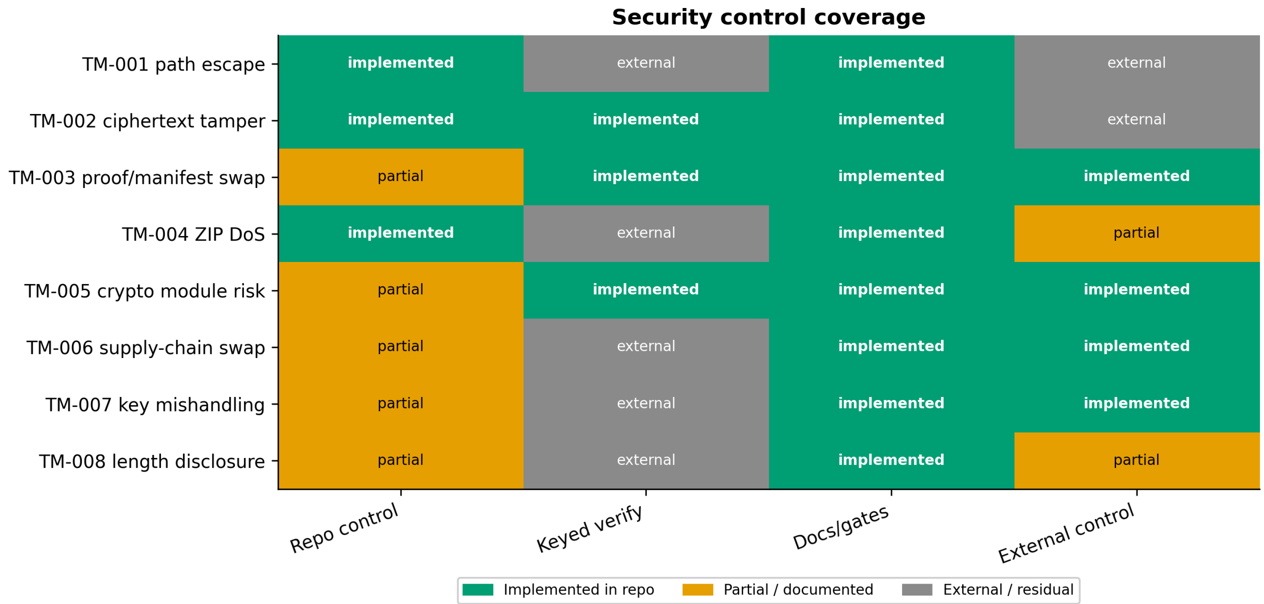

Figure 11: Security-control matrix mapping ENTO threat IDs to repository-enforced, partial, and external controls. Implemented cells are backed by code/tests/docs in this repository; external cells require deployment environment controls such as artifact signing, HSM/KMS key custody, or SIEM routing. Filters: all benchmark rows. Data from `ento_benchmark_results.csv`. Generated by `generate_security_control_matrix_figure` in `src/figures.py`.

7 Formal model

This section gives a compact formal account of the ENTO container so that the benchmark results in sec. 8 and sec. 9 can be read against precise definitions rather than prose. The model is deliberately minimal: it fixes notation for the track envelope, states the ciphertext-expansion law that the measurements verify exactly, and defines the integrity and observability predicates that sec. 6 and sec. 5 exercise.

7.1 Container as a typed track map

Fix a master key $K \in \{0,1\}^{8\kappa}$ with $\kappa = 32$ bytes. A container C over a finite track-id set $\mathcal{I}$ is a partial map from track ids to typed, encrypted payloads,

$$C : \mathcal{I} \rightharpoonup \mathcal{U} \times \mathcal{B}, \tag{1}$$

where $\mathcal{U}$ is the ontology-URI set (the *typed* axis of the format) and $\mathcal{B} = \{0,1\}^*$ is the byte space of stored ciphertext. The *omnitrack* property of eq. 1 is that $|\mathcal{I}|$ is unbounded and the tracks are heterogeneous: a single C may bind `ento:timeseries.eeg`, `ento:genomics.vcf`, and `ento:spectrogram` simultaneously (sec. 4).

7.2 Per-track key derivation and encryption

Each track is encrypted under its own key, derived from K by HKDF-SHA256 [Krawczyk and Eronen, 2010, National Institute of Standards and Technology, 2015] with a track-binding `info` string,

$$k_i = \mathrm{HKDF\text{-}SHA256}(K, \texttt{"ento:track:"} \,\|\, i), \qquad i \in \mathcal{I}, \tag{2}$$

so distinct tracks receive cryptographically independent keys and the track id is bound into the derivation (eq. 2). Operationally, this means a ciphertext for `eeg` is not merely a blob under the master key; it is checked under the `eeg` subkey, so a cross-track swap fails before plaintext is released. For a plaintext $m_i \in \{0,1\}^{8n}$ of n bytes, the stored member is the concatenation

$$\mathrm{ENC}(k_i, m_i) = \nu_i \,\|\, \tau_i \,\|\, c_i, \tag{3}$$

where ν_i is the 12-byte AES-256-GCM nonce, τ_i is the 16-byte authentication tag, and c_i is the ciphertext body [National Institute of Standards and Technology, 2001, McGrew, 2008, Dworkin, 2007, McGrew and Viega, 2004]. Because GCM is a stream-based AEAD mode, it preserves the length of the bytes supplied to encryption; for default format 0.4.0, those bytes are the original-length prefix plus payload padded to a PADMÉ bucket.

7.3 The ciphertext-expansion law

Let $H = 28$ be the fixed header size (nonce plus tag, from eq. 3). Since tracks are stored uncompressed (`ZIP_STORED`) and format 0.4.0 pads the encrypted body to PADME$(n + 8)$ bytes, the default per-track expansion ratio is exactly

$$r(n) \;=\; \frac{H + \mathrm{PADME}(n+8)}{n}. \tag{4}$$

eq. 4 is an identity, not an empirical fit: it predicts a strictly decreasing upper envelope in n that asymptotes toward 1 as $n \to \infty$, with overhead attributable to the authenticated header and PADMÉ bucket. fig. 20 overlays the measured fixture-track ratios on this curve; the maximum absolute residual is reported in the figure title and is at floating-point noise level, confirming the law holds for the implementation rather than merely approximating it. The decomposition of header and padded body parts is shown per track in fig. 5.

7.4 Integrity predicate

Verification reports an `integrity` level (see sec. 6) that is a function of what was actually checked. Writing key for "master key supplied and every track's GCM tag verified", dig for "every ciphertext digest present and matched", the level is

$$\mathrm{integrity}(C) = \begin{cases} \textsf{key-authenticated} & \text{if } \textsf{key}, \\ \textsf{digest-only} & \text{if } \neg\,\textsf{key} \wedge \textsf{dig}, \\ \textsf{unverified} & \text{otherwise.} \end{cases} \tag{5}$$

Only the key-authenticated branch of eq. 5 resists an adversary who controls the container bytes: the unkeyed digests and proof chain (sec. 5) are recomputable by anyone, so digest-only detects *accidental* corruption alone. This distinction is why the manuscript reports keyless verification as a pipeline consistency check and tamper rejection as a keyed AEAD result. The proof hash is over the implementation's exact emitted manifest JSON bytes, not a general JCS serialization [Rundgren et al., 2020]. The CLI fails closed on unverified by default (sec. 6).

7.5 Observability redaction is monotone

Let $\mu_\ell(C)$ be the exported manifest at observability level $\ell \in \{0, \dots, 3\}$, and $|\mu_\ell(C)|$ its byte size. Redaction is *subtractive on field content*: lowering ℓ only removes or shortens field classes (digests, then resolution descriptors, then type URIs — the sealed level replaces each type URI with the shorter sentinel `ento:opaque`), and never re-encrypts the payload. Under the implementation's URI registry — where every ontology URI is at least as long as that sentinel — the exported manifest size is therefore observed to be monotone non-decreasing in the level,

$$\ell_1 \le \ell_2 \implies |\mu_{\ell_1}(C)| \le |\mu_{\ell_2}(C)|. \tag{6}$$

eq. 6 is a property of the current field schema rather than a guarantee for arbitrary registries (a hypothetical type URI shorter than the sentinel could invert the top step); it is what fig. 16 and fig. 3 show across the fixture tracks, and it is the basis for the confidentiality-versus-auditability trade-off discussed in sec. 9.

8 Results

Benchmarks run via `scripts/ento_analysis.py` and hydrate into this section through `scripts/z_generate_manuscript_variables.py`.

8.1 Code-derived figures

All plots are generated from `output/data/ento_benchmark_results.csv` — never hand-edited. The pipeline in `src/analysis.py`:

1. Runs `src/benchmarks.py::run_all_benchmarks` using parameters from `manuscript/config.yaml` (`experiment.benchmark_repetitions`, `observability_levels`, `medium_track_bytes`).
2. Calls `src/figure_registry.py::generate_all_figures`, which dispatches each registered generator in `src/figures.py`.
3. Writes `../figures/figure_registry.json` with `generated_by` paths, CSV provenance, `kind`, `manuscript_section`, and `caption_token` names that mirror the manuscript figure-caption variables.

The manuscript references 21 figures at 300 DPI via `experiment.viz` in config. Registry path after analysis: `../figures/figure_registry.json`. Alt text for each figure is injected from `FIGURE_SPECS[].caption` in `src/figure_registry.py` (see `docs/figure_registry.md`).

8.2 Benchmark overview

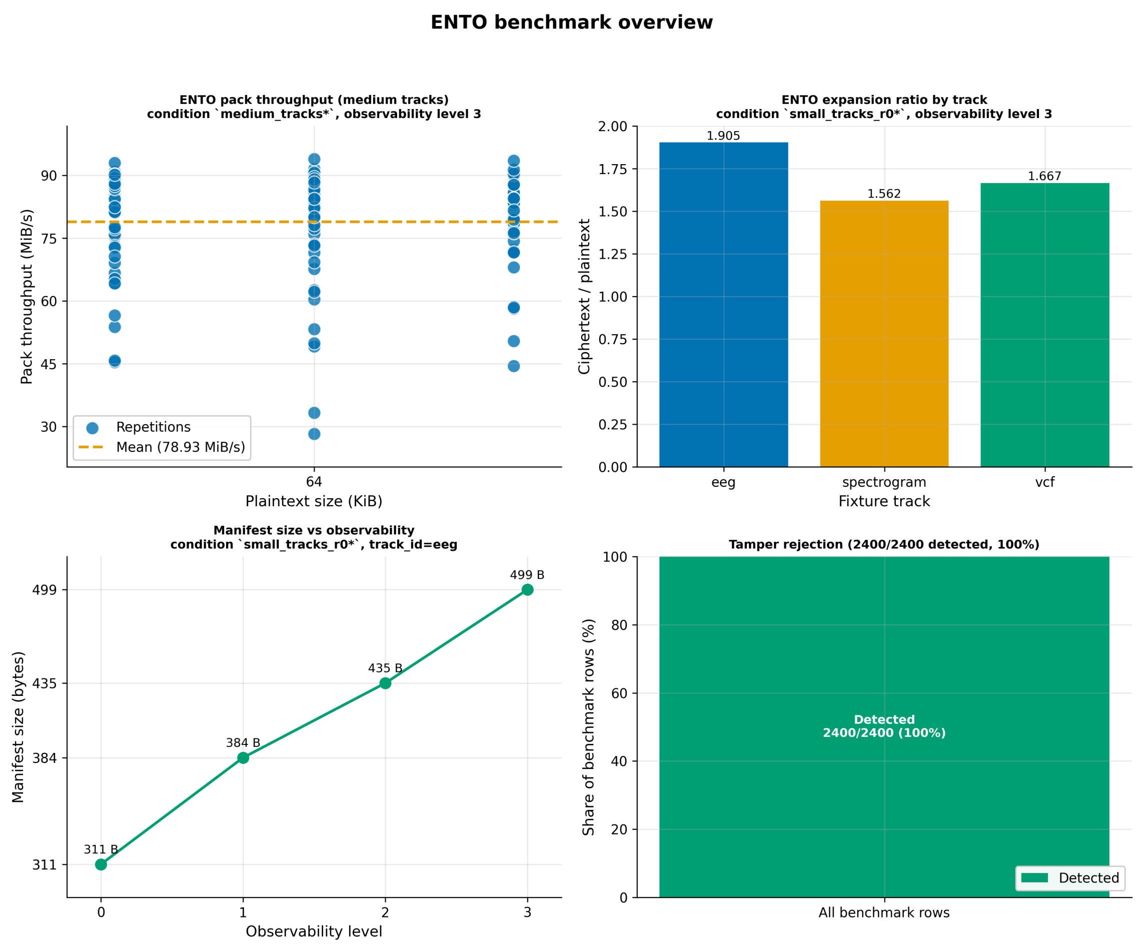

Figure 12: At-a-glance 2x2 summary of the four headline benchmark views, each reusing its standalone figure's filters: pack throughput against plaintext size (top-left), ciphertext expansion ratio by fixture track (top-right), EEG manifest size against observability level (bottom-left), and tamper-detection outcomes (bottom-right). Read together they show the local evidence for authenticated confidentiality, graded observability, format overhead, and timing dispersion. See the standalone figures for print-scale detail. Filters: condition `medium_tracks*`, observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_benchmark_overview_figure` in `src/figures.py`.

fig. 12 summarizes the four primary benchmark views in one panel: medium-track throughput, fixture expansion ratios, EEG manifest shrinkage across observability levels, and tamper-detection outcomes. Use the standalone figures below for print-scale detail.

8.3 Throughput

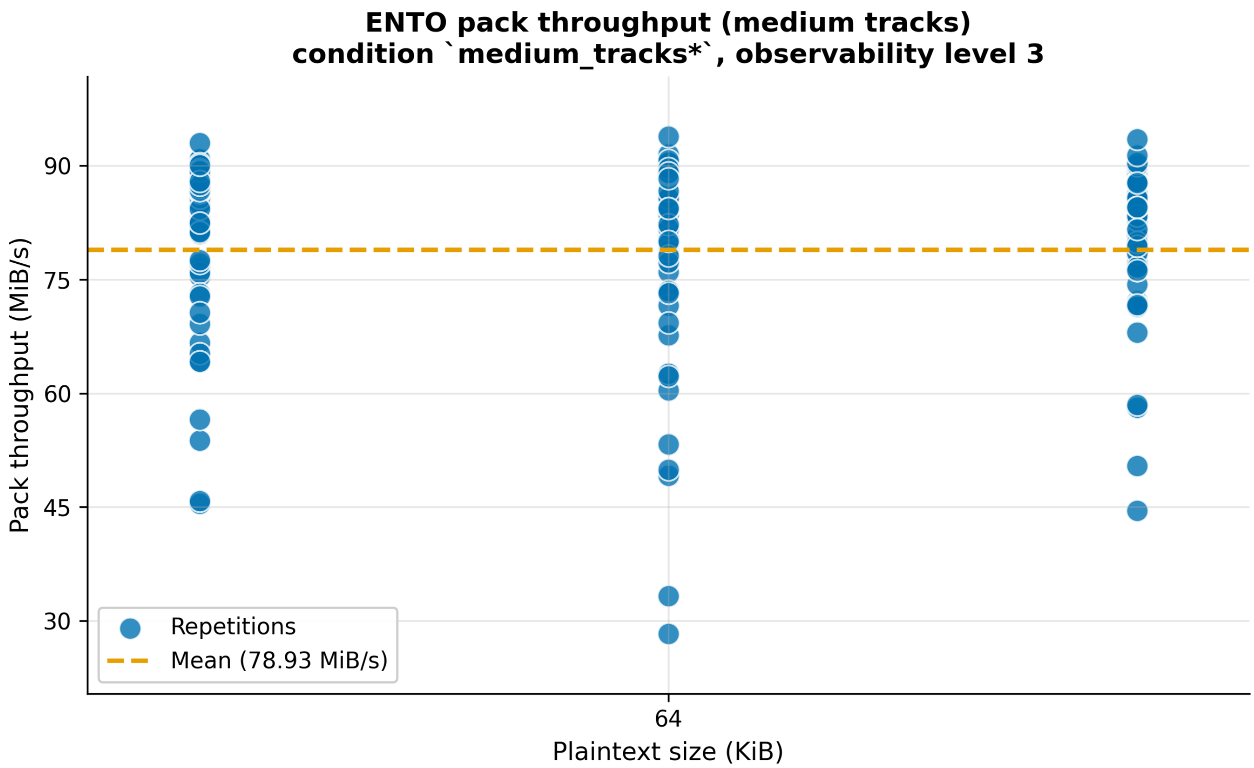

Figure 13: Pack throughput, in MiB/s, against plaintext size for the medium synthetic-track condition. Each marker is one repetition (jittered horizontally to separate coincident points); the dashed line marks the mean across repetitions. This is a local wall-clock snapshot: it reports measured dispersion for the configured release run and does not assert cross-host or cross-implementation throughput superiority. Filters: condition `medium_tracks*`, observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_throughput_figure` in `src/figures.py`.

fig. 13 summarizes pack throughput for the medium-track condition (65536 bytes plaintext) at observability level 3. Mean throughput across filtered repetitions: 78.9296 MiB/s. Registry filters: `condition_prefix=medium_tracks`, `observability_level=3`. The plot overlays individual repetitions with a dashed mean line.

8.4 Expansion ratio

fig. 14 compares ciphertext expansion on fixture tracks for `small_tracks_r0` at observability level 3. Mean expansion ratio across those rows: 1.7113. Registry filters: `condition_prefix=small_tracks_r0`, `observability_level=3`.

8.5 Expansion heatmap

fig. 15 shows mean `expansion_ratio` for every `condition` × `track_id` pair at observability level 3, highlighting how fixture size and synthetic medium tracks diverge in ciphertext overhead.

8.6 Observability manifest size

fig. 16 traces manifest payload size for the EEG fixture across observability levels declared in config (0,1,2,3). Registry filters: `condition_prefix=small_tracks_r0`, `track_id=eeg`. At level 3, manifest size for this track averages 499 bytes in the benchmark CSV.

8.7 Summary table

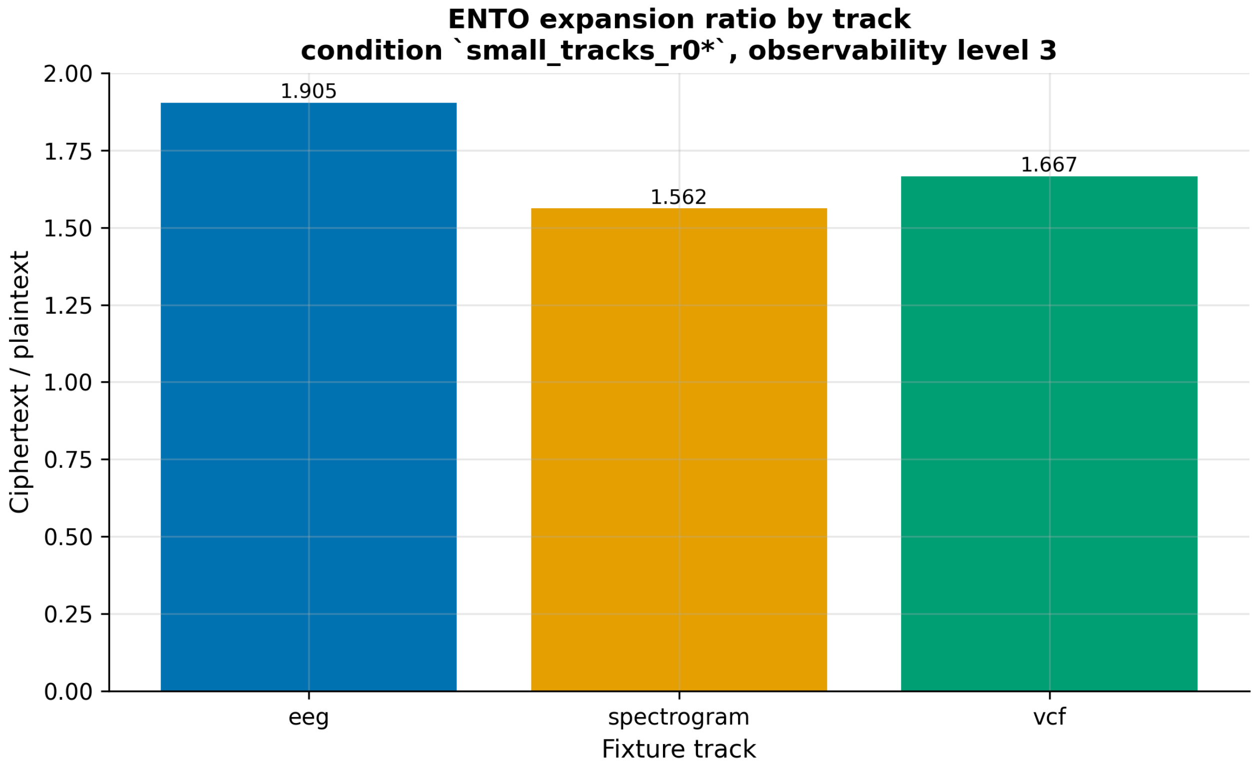

Figure 14: Ciphertext-to-plaintext expansion ratio per fixture track, with the exact ratio printed above each bar. Expansion combines the 28-byte per-track AEAD header (nonce plus authentication tag) with the version-selected ciphertext body; the default 0.4.0 profile PADME-pads that body, so small tracks pay proportionally more overhead while larger tracks approach a ratio of one. Filters: condition `small_tracks_r0*`, observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_expansion_figure` in `src/figures.py`.

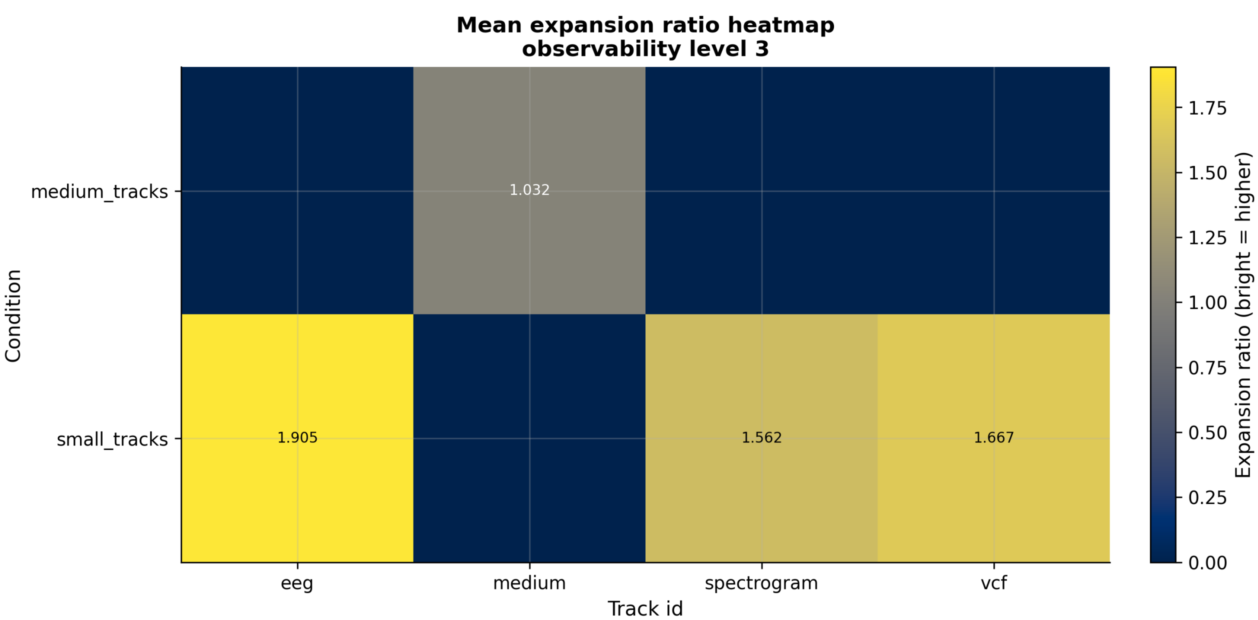

Figure 15: Mean ciphertext expansion ratio for each base benchmark condition (rows, with repetition suffixes collapsed) crossed with every track id (columns), rendered on the colorblind-safe cividis map where brighter (yellow) cells mark higher overhead and darker (blue) cells mark lower overhead. The exact ratio is overlaid on each cell. Small fixture tracks light up; the large synthetic medium track stays dark, isolating fixed header cost from payload size at a glance. Filters: observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_expansion_heatmap_figure` in `src/figures.py`.

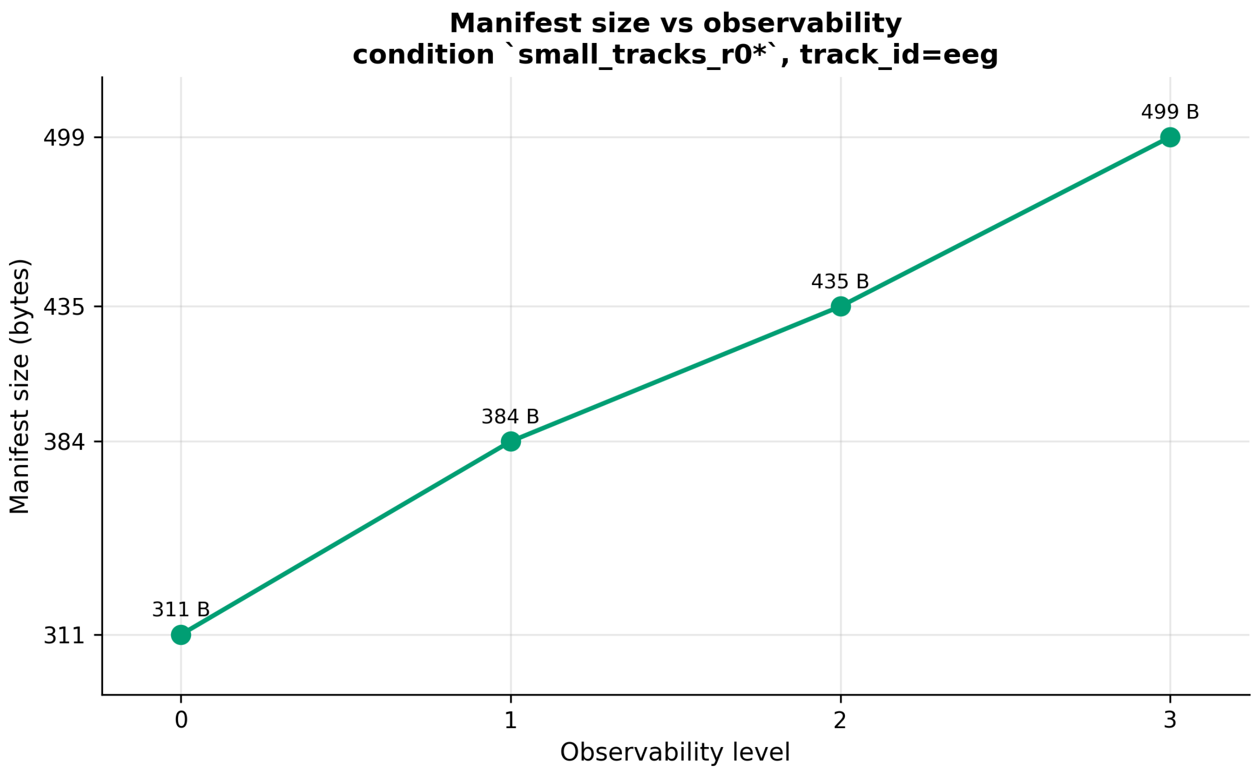

Figure 16: Exported manifest size, in bytes, against observability level for the EEG fixture, with each point labelled by its byte count. Size falls monotonically as the export level drops from auditable to sealed: each step removes a field class (plaintext digests, then resolution descriptors, then type URIs) without re-encrypting the payload, making the confidentiality-versus-auditability trade-off directly measurable. Filters: condition `small_tracks_r0*`, track_id=eeg. Data from `ento_benchmark_results.csv`. Generated by `generate_observability_figure` in `src/figures.py`.

Track	Type	Plaintext bytes	Expansion	Throughput (MiB/s)
eeg	ento:timeseries.eeg	42	1.904762	0.060792
spectrogram	ento:spectrogram	64	1.562500	0.076481
vcf	ento:genomics.vcf	60	1.666667	0.086387

9 Benchmark interpretation

Benchmarks execute via `src/benchmarks.py::run_all_benchmarks` and write `output/data/ento_benchmark_results.csv` (SHA-256: `47f1e923ae809ab5e384e1d5d01b48f47de62afe04092f9922300de2a163672a`). Filter definitions for plots and injected statistics match `docs/methods.md`. This section interprets primary metrics declared in `experiment_plan.yaml`.

9.1 Baselines and conditions

Condition	Role	Tracks
`small_tracks_r*`	Baseline	Fixture tracks under 4 KiB
`medium_tracks_r*`	Throughput	Synthetic 65536-byte spectrogram payload
Observability sweep	Ablation	Export levels 0,1,2,3

Each row records pack/unpack latency, expansion ratio, manifest bytes, and a tamper-injection outcome.

9.2 Pack and unpack latency

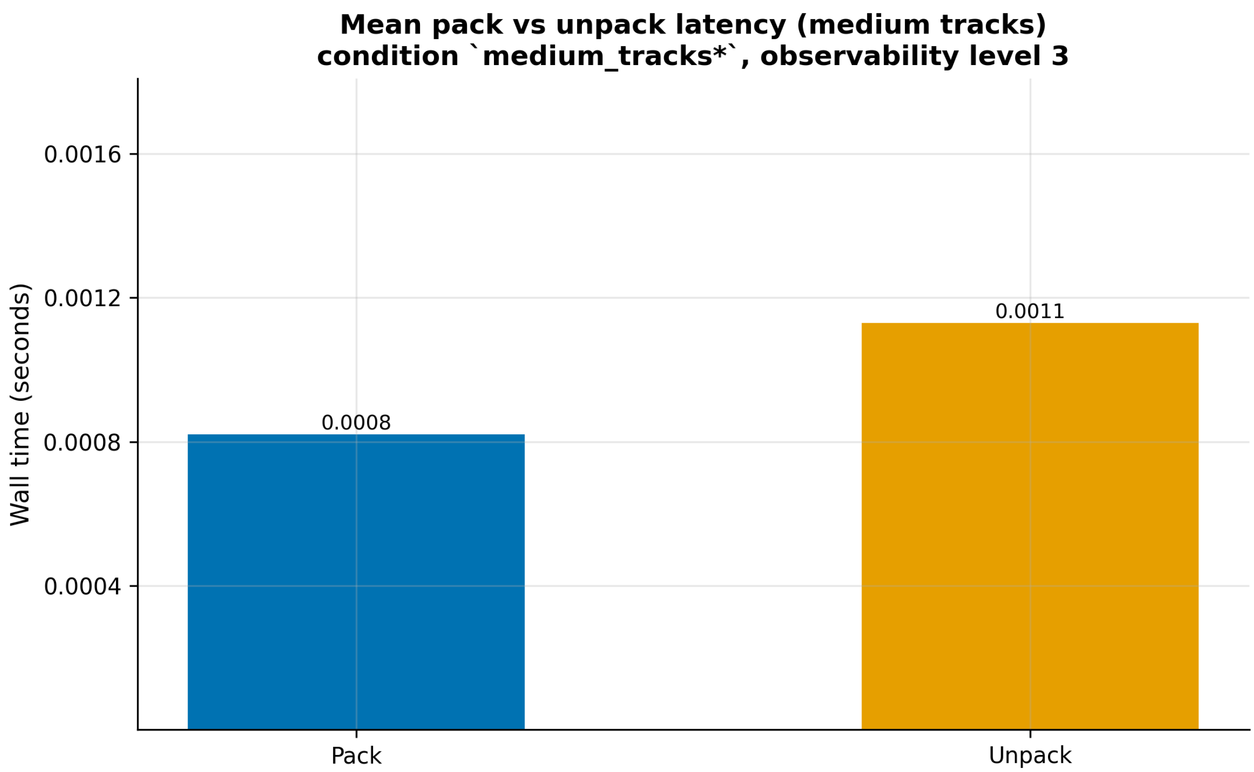

Figure 17: Mean pack and unpack wall-clock time, in seconds, for the medium-track condition, with the exact time printed above each bar. In this local run, unpack carries the additional cost of authenticating the AEAD tag and checking plaintext digests before any bytes are released, so it is the more expensive half of the measured round trip — the price of verify-before-use. Filters: condition `medium_tracks*`, observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_unpack_latency_figure` in `src/figures.py`.

fig. 17 compares mean pack versus unpack wall time on medium tracks at observability level 3. Mean unpack latency on the throughput condition: 0.001130 s (fig. 13).

9.3 Throughput across observability levels

fig. 18 plots pack throughput for all `medium_tracks_*` rows grouped by observability level, with a min–max band across repetitions. Lower export levels reduce manifest work during pack, which can shift throughput relative to level 3.

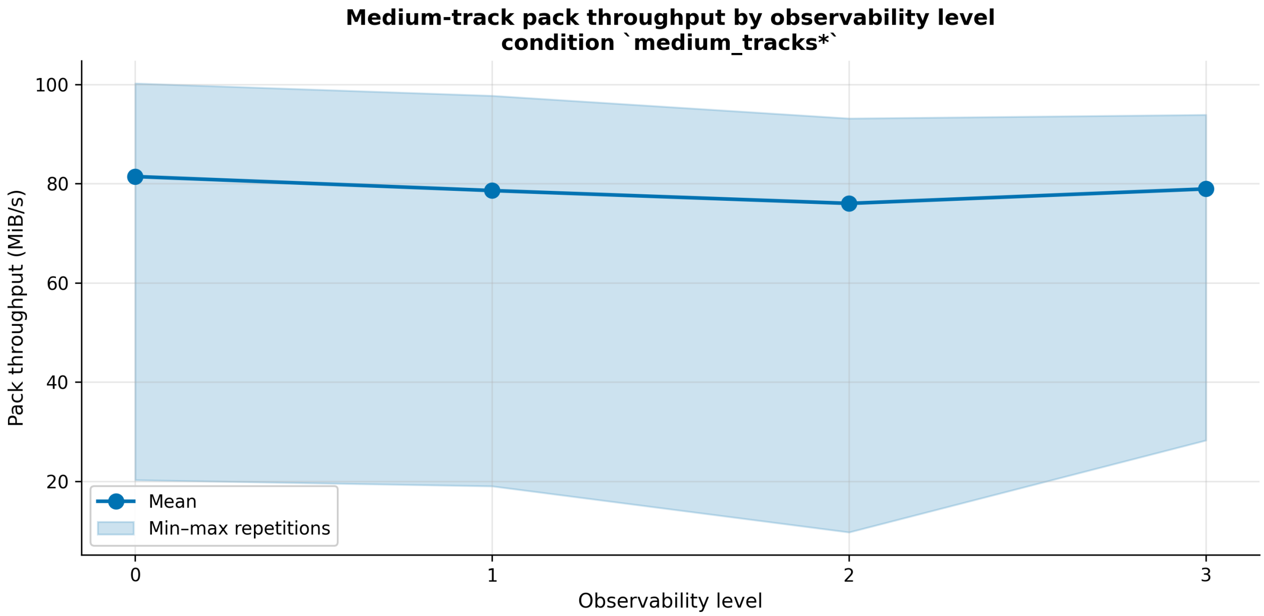

Figure 18: Pack throughput, in MiB/s, against observability level for the medium-track condition: the solid line is the per-level mean and the shaded band spans the min–max across repetitions. Throughput is largely flat across levels in this local run. Redaction trims manifest fields at export time and never touches the encrypted payload, consistent with observability being a metadata-only control. Filters: condition `medium_tracks*`. Data from `ento_benchmark_results.csv`. Generated by `generate_throughput_by_observability_figure` in `src/figures.py`.

9.4 Manifest size vs throughput trade-off

fig. 19 scatters `manifest_bytes` against `pack_throughput_mib_s` for medium tracks at level 3, linking observability export size to pack performance (sec. 5).

9.5 Primary metrics

Pack throughput (MiB/s) measures plaintext bytes divided by pack wall time on the medium-track condition at observability level 3. Mean across 150 repetitions: 78.9296 MiB/s (SD 12.0567, CV 15.3%; fig. 13, dispersion in fig. 21). The mean alone is a weak summary here — see the statistical-dispersion subsection below.
Unpack latency on the same condition averages 0.001130 seconds per container. Unpack authenticates AEAD tags (format 0.4.0) before release and checks plaintext SHA-256 digests when present.
Expansion ratio compares ciphertext track size to plaintext on fixture tracks. Mean: 1.7113 (fig. 14). Overhead reflects the 28-byte GCM header (`nonce || tag`) plus the version-selected ciphertext body; for format 0.4.0, that body is PADMÉ-padded (fig. 5).

9.5.1 Expansion follows a closed-form law

Expansion is not an empirical artifact to be averaged — it is fixed by the selected format. For format 0.4.0, a track of n plaintext bytes occupies the 28-byte header plus a PADMÉ bucket containing an original-length prefix and payload, giving r(n) = (H + PADME(n + 8)) / n as derived in sec. 7. fig. 20 overlays the measured fixture-track ratios on this analytic curve: every point lands on the model (maximum absolute residual at floating-point noise, reported in the figure). Because the header and padding function are spec-fixed rather than fit to the data, the overlay is an empirical confirmation that the implementation realises the closed form over the sampled sizes — not a regression with free parameters. The practical reading is that overhead combines a constant header tax with a bounded length-hiding bucket; it is predictable before packing, but it should not be described as pure timing or compression behavior.

9.6 Statistical dispersion and reliability

The release benchmark runs 150 repetitions, a 50x increase over the 3-repetition pilot setting used for routine smoke checks. Each repetition contributes 16 rows, so the expected release matrix is 2400 rows before any validation filtering. For medium-track pack throughput at observability level 3 the sample is n = 150 (df = 149), with mean 78.9296 MiB/s, sample standard

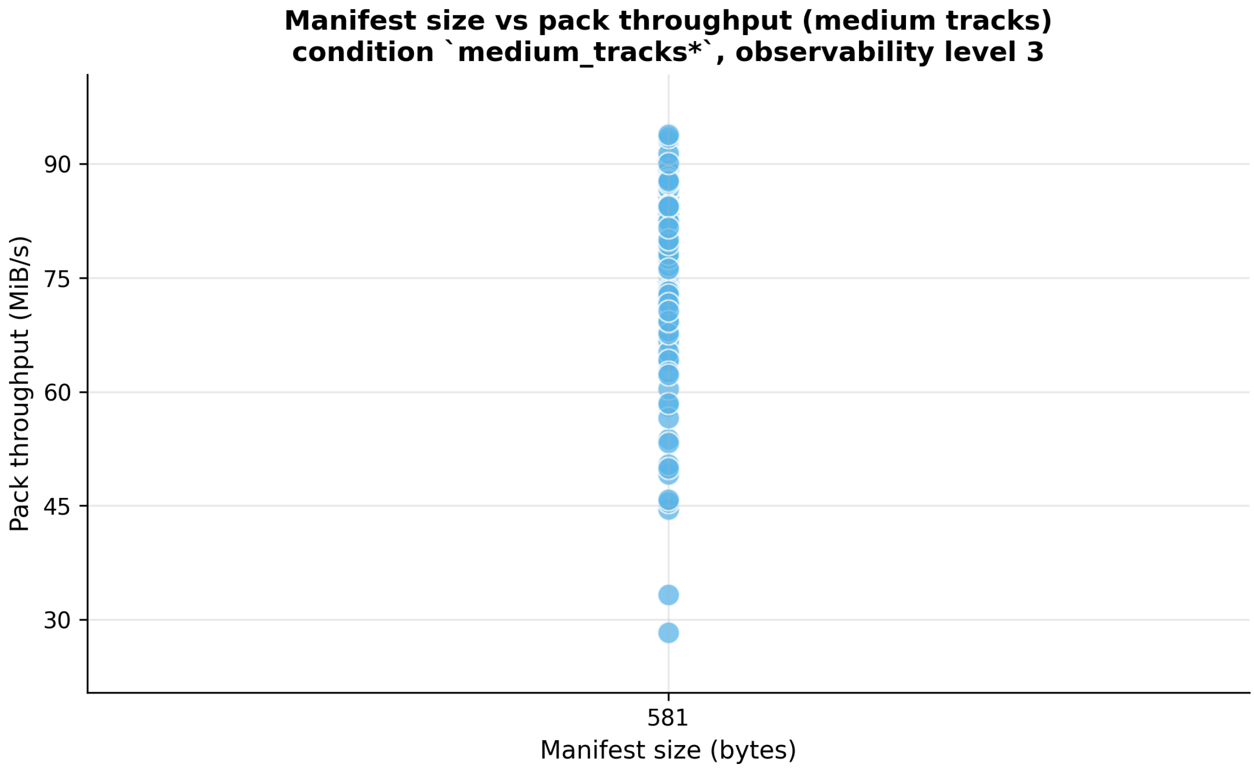

Figure 19: Exported manifest size, in bytes, against pack throughput, in MiB/s, for the medium-track condition — one point per benchmark row. In this local run, the panel does not show a visible downward trend, so the generated matrix treats manifest size and payload throughput as empirically decoupled rather than as a portable performance law. Filters: condition `medium_tracks*`, observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_observability_tradeoff_figure` in `src/figures.py`.

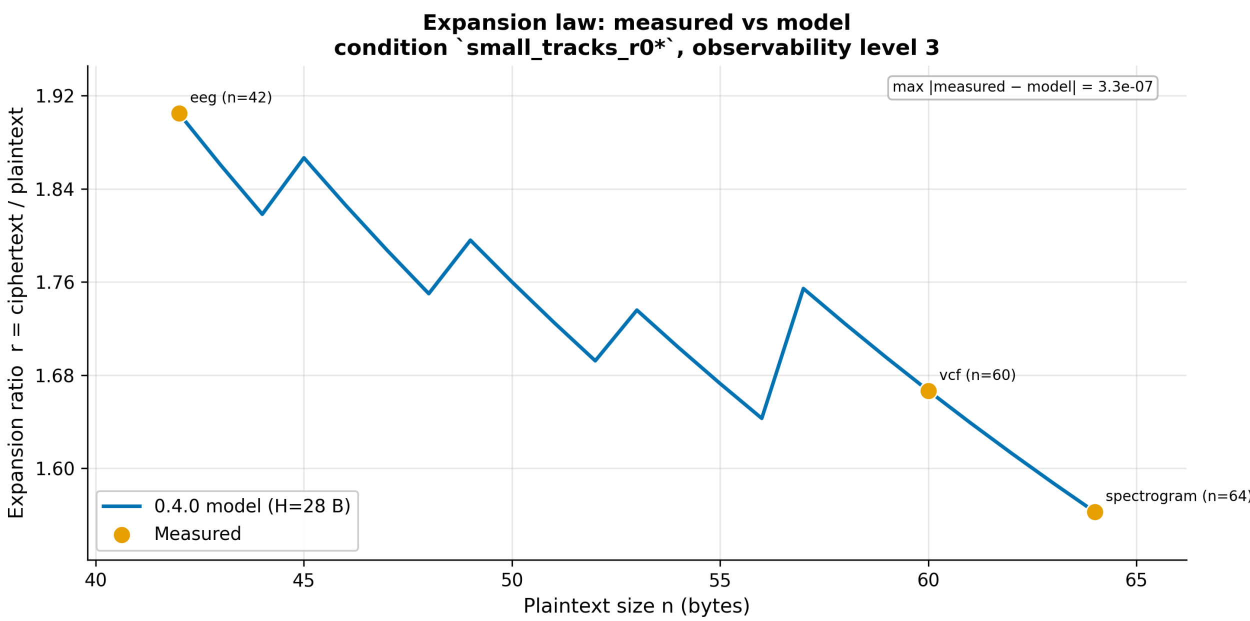

Figure 20: Measured ciphertext expansion ratio (markers, one per fixture track) overlaid on the version-aware model. For the default 0.4.0 profile, r(n) = (H + PADME(n + 8)) / n because the encrypted body carries an eight-byte original-length prefix before PADME bucketing; unpadded compatibility formats reduce to r(n) = (H + n) / n. The maximum absolute residual is reported as a fidelity badge, confirming a spec-fixed identity rather than a statistical fit. Filters: condition `small_tracks_r0*`, observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_expansion_law_figure` in `src/figures.py`.

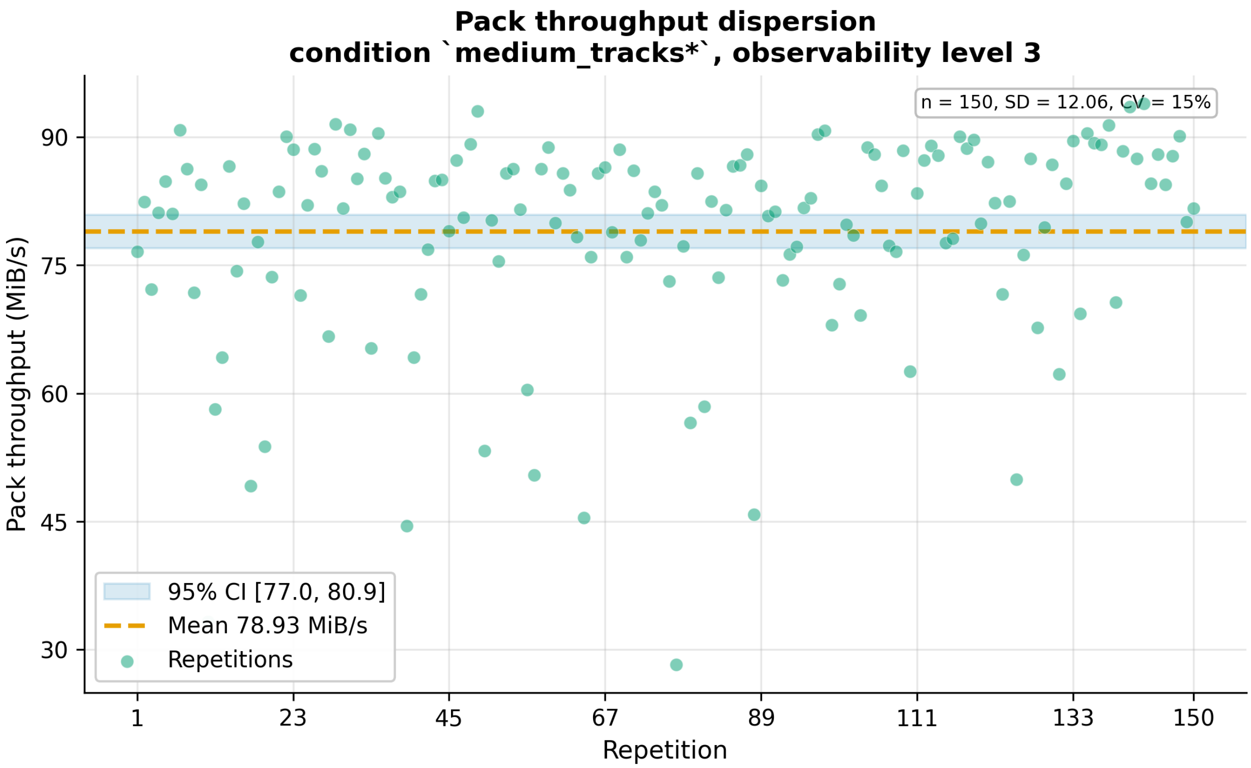

Figure 21: Per-repetition pack throughput for the medium-track condition with the mean (dashed) and a two-sided 95% Student-t confidence interval (shaded band; t critical value for n-1 degrees of freedom, tabulated for small df and expanded for large df without scipy). Each marker is one repetition, drawn with sparse x-axis ticks for dense release runs; the inset reports token-derived n, sample standard deviation, and coefficient of variation. The band is the honest counterpart to the exact expansion law: wall-clock throughput carries host-specific measurement noise, so the mean alone would overstate precision. Filters: condition `medium_tracks*`, observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_throughput_dispersion_figure` in `src/figures.py`.

deviation 12.0567 MiB/s, and coefficient of variation 15.3% (fig. 21). The two-sided 95% confidence interval is [76.9843, 80.8748] MiB/s, computed as two-sided 95% Student-t interval using a no-SciPy large-df expansion for t(149, 0.975).
The larger repetition count improves the within-run estimate, but it does not turn a local wall-clock measurement into a portable performance claim. Re-running the benchmark re-measures elapsed time, so the mean, CV, and interval bounds remain host-state snapshots affected by CPU scheduling, filesystem cache state, background load, and Python/cryptography build details. The operational claim is therefore bounded: **the reported throughput summarizes this release run, and ENTO makes no cross-implementation throughput-superiority claim from it**. This caveat is specific to wall-clock metrics. The data-derived metrics (expansion ratio, ciphertext byte counts, manifest size, and tamper outcomes) are exact functions of byte counts, schema choices, or deterministic checks; their reproducibility is anchored separately by the data fingerprint. The figures split accordingly: fig. 20 is a byte-exact overlay, fig. 21 is a re-measured timing sample, and fig. 22 shows which columns belong to each side of the boundary. The code enforces the distinction through the figure registry's `data_derived` determinism contract.

9.6.1 Variation by metric

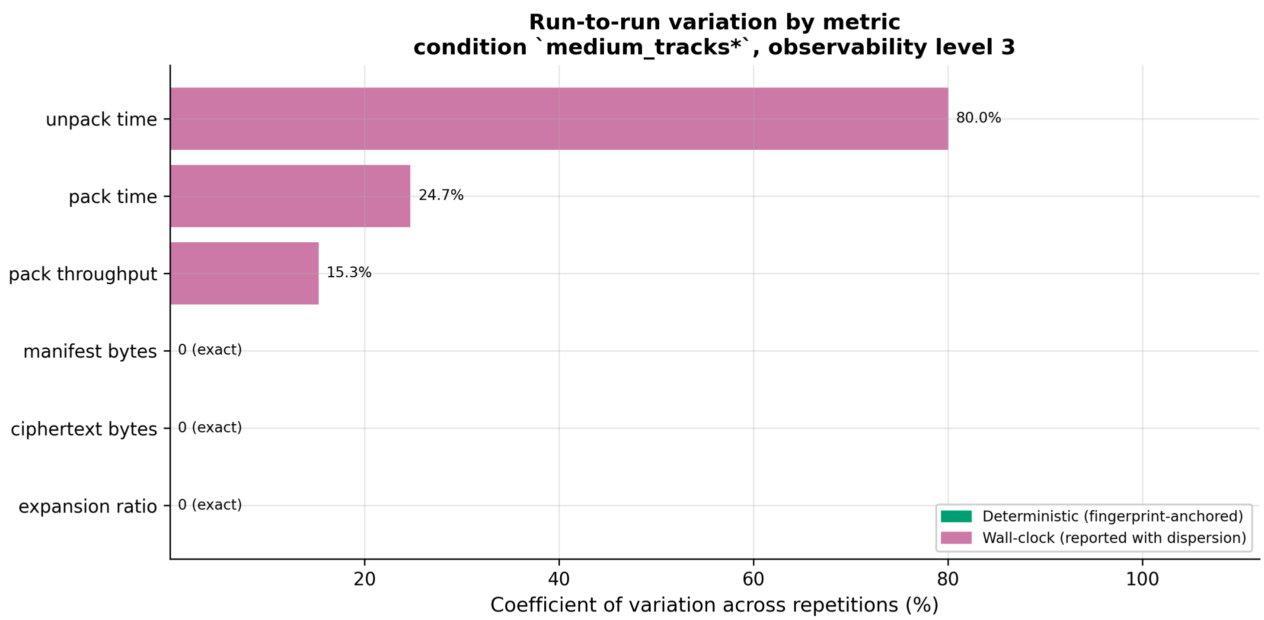

Figure 22: Coefficient of variation (CV), in percent, of each benchmark metric across the repetitions of the medium-track condition, drawn as a horizontal bar per metric. The data-derived metrics — expansion ratio, ciphertext bytes, and manifest bytes — sit at exactly zero because they are fixed by the format and the manifest schema, while the wall-clock metrics — pack time, unpack time, and pack throughput — carry real run-to-run dispersion. This is the visual justification for the data fingerprint: it anchors only the zero-CV (deterministic) columns and reports the timing columns with their dispersion rather than hashing them. Filters: condition `medium_tracks*`, observability level 3. Data from `ento_benchmark_results.csv`. Generated by `generate_determinism_cv_figure` in `src/figures.py`.

fig. 22 makes the deterministic/measured split directly visible: the coefficient of variation across repetitions is exactly zero for the data-derived columns (expansion ratio, ciphertext bytes, manifest bytes) and strictly positive for the wall-clock columns (pack and unpack time, pack throughput). This is the empirical basis for the data fingerprint in sec. 12, which hashes only the zero-variation columns and leaves the timing columns to be reported with their dispersion (fig. 21) rather than folded into a reproducibility anchor.

9.7 Observability trade-off

Manifest byte counts shrink monotonically from level 3 $\rightarrow$ 0 (fig. 16; numeric levels in sec. 5). Level 0 minimizes metadata leakage at the cost of auditability—appropriate for sealed distribution; level 3 supports reproducibility checks against fixture digests in sec. 4.

9.8 Tamper detection

Every benchmark row corrupts a ciphertext tag byte and expects unpack to fail closed. Detected attempts: 2400 of 2400 (fig. 6). Validation requires tamper detection rate 1.0 with status pass in `output/reports/benchmark_validation.json`, and `container_verification.json` with aggregate status true (sec. 6).

9.9 Results table

Table tbl. 1 summarizes fixture-track rows at observability level 3:

Table 1: Benchmark summary for fixture tracks at observability level 3.

Track	Type	Plaintext bytes	Expansion	Throughput (MiB/s)
eeg	ento:timeseries.eeg	42	1.904762	0.060792
spectrogram	ento:spectrogram	64	1.562500	0.076481
vcf	ento:genomics.vcf	60	1.666667	0.086387

Raw CSV rows: 2400 total across all conditions and repetitions (150 repetitions $\times$ 16 rows per repetition; expected total 2400).

10 Conclusion

ENTO demonstrates a narrow but useful point: a flat ZIP container [zip, 2024] can carry typed multimodal tracks, authenticated encryption on format 0.4.0, graded observability, and proof export without becoming a repository, policy engine, or hosted service. The benchmark pipeline integrates with registry-backed manuscript variables and claim-ledger tests so manuscript prose tracks measured outputs rather than stale summaries.

10.1 Contributions

1. **Typed track ontology** — URI registry (`ento:timeseries.eeg`, `ento:genomics.vcf`, `ento:spectrogram`) with schema-validated resolution descriptors (sec. 4).
2. **Per-track authenticated envelope** — Fixed header (`nonce || tag || ciphertext`) with HKDF-derived keys [Krawczyk and Eronen, 2010]; AES-256-GCM on format 0.4.0 (sec. 3).
3. **Observability levels 0–3** — Export-time manifest redaction without re-encryption (sec. 5).
4. **Proof export binding** — `manifest_sha256` chained to track digests for anchoring (sec. 5).
5. **Reproducible benchmarks** — 2400 CSV rows with 2400 tamper detections at rate 1.0, wired to registry-backed manuscript variables and claim ledger tests.
6. **Security verification gate** — `container_verification.json`, structured CLI verify logging, and nation-state deployment checklist (sec. 6).
7. **Honest verification and a hardened format ladder** — an `integrity` contract that reports key-authenticated versus keyless corruption-detection and fails closed by default, a default 0.4.0 profile with associated-data binding and PADMÉ length-padding [Nikitin et al., 2019], and compatibility formats 0.2.0, 0.3.0 and 0.3.1 that remain readable beside it (sec. 14).

The remaining work is deliberately outside the container core: streaming partial decrypt, public-release artifact signing/provenance, KMS/HSM adapters, nonce-misuse-resistant future formats, and formal interoperability tests against HDF5 [hdf, 2024] and RO-Crate [roc, 2024]. Those additions can be layered around the 0.4.0 ciphertext contract documented in `data/ento_track_header.ksy`, with supply-chain and key-custody controls following external guidance rather than being implied by the ZIP envelope itself [National Institute of Standards and Technology, 2020, 2022, Torres-Arias et al., 2019, Gueron and Lindell, 2019]. The planned public destination is `https://github.com/docxology/entofile`; the release candidate remains reproducible from the current `projects/working/entofile` tree until promotion.

11 Experimental setup

11.1 Software

- Python 3.12.13 on macOS-26.5.1-arm64-arm-64bit
- ENTO format version 0.4.0 (suite aes-256-gcm)
- Manuscript config version 0.4 (paper version 0.4)

11.2 Fixtures

File	Track id	Ontology type
`data/fixtures/eeg.csv`	eeg	`ento:timeseries.eeg`
`data/fixtures/sample.vcf`	vcf	`ento:genomics.vcf`
`data/fixtures/spectrogram.bin`	spectrogram	`ento:spectrogram`

Evidence classes are intentionally separated rather than collapsed into a vague real-world-input claim. Fixture inputs are **committed deterministic fixture inputs**; the medium throughput condition is a **generated synthetic throughput stress track** of 65536 bytes; conformance cases are **deterministic test-vector containers**; and the benchmark CSV, reports, figures, and rendered manuscript are **real ZIP, crypto, filesystem, and render execution outputs**. The fixture and conformance bytes are therefore reproducibility inputs, not field-collected research observations. This separation follows reproducible computational research practice: expose scripts, data classes, run parameters, and outputs so readers can inspect what was actually executed [Sandve et al., 2013, Wilson et al., 2017]. Figure export uses 300 DPI from `experiment.viz` in config.

11.3 Benchmark protocol

- Repetitions: 150
- Rows per repetition: 16 (expected total 2400)
- Observability sweep: 0,1,2,3 (maximum level 3)
- Analysis entry point: `uv run python scripts/ento_analysis.py`
- CLI: `uv run python scripts/ento_cli.py pack|unpack|inspect|proof|genkey`

Each repetition packs fixture tracks at every observability level, packs the synthetic medium track for throughput, corrupts one ciphertext tag byte, and records whether unpack rejects the container. Results append to `output/data/ento_benchmark_results.csv` with columns for pack/unpack seconds, expansion ratio, manifest bytes, and tamper outcome. This is a real execution of the ENTO pack/unpack/verify path over documented inputs, not a mock or a substituted result table.
Validation requires `output/reports/benchmark_validation.json` to report status pass and tamper detection rate 1.0, and `output/reports/container_verification.json` to report all benchmark samples verified (`true`). HKDF and GCM vectors in `data/test_vectors/` are exercised before release. Figure DPI follows `src/figures.py::VIZ_CONFIG`; caption tokens follow `src/figure_registry.py` (`tests/test_figure_captions.py`). Claims bind via `data/claim_ledger.yaml` and `docs/claim_ledger.md`.

11.4 Statistical methods

Each timing metric is summarized across the 150 repetitions as mean, sample standard deviation $(n - 1)$, coefficient of variation, and a two-sided 95% confidence interval (`src/benchmark_stats.py::summary_stats`). For the headline throughput sample this is two-sided 95% Student-t interval using a no-SciPy large-df expansion for $t(149, 0.975)$, with $n = 150$ and $df = 149$. Wall-clock metrics carry run-to-run measurement noise, so the interval describes this release run under local host conditions rather than a cross-host population result — see fig. 21 and sec. 9. Data-derived metrics (expansion ratio, manifest size, ciphertext byte counts, tamper outcomes) are exact functions of byte counts, schema choices, or deterministic checks; their reproducibility is anchored by `BENCHMARK_DATA_FINGERPRINT`, not by timing dispersion. The figure registry encodes this split as a per-figure `data_derived` determinism contract, enforced by `tests/test_figure_determinism.py`.
The project does not request an external artifact badge in this release, but the generated PDF/HTML, source-bound variables, figure registry, and release manifest are organized to make the artifact-review distinction between available, functional, and reusable evidence explicit [Association for Computing Machinery, 2024].

11.5 Environment notes

Benchmarks use a fixed master key per run generated in `run_all_benchmarks`. Timing uses `time.perf_counter()` around pack and unpack calls; throughput divides plaintext MiB by pack seconds on the medium-track rows at observability level 3 only.

12 Reproducibility

Configuration hash: `68cac678fc92a913` (SHA-256 prefix of `manuscript/config.yaml`, version 0.4).
Author: Daniel Ari Friedman. Keywords: research data formats, authenticated encryption, AES-256-GCM, reproducible research, multimodal containers, metadata leakage, observability levels, FAIR data packaging.

12.1 Regenerate artifacts

```
uv run python scripts/ento_analysis.py
uv run python scripts/build_dashboard.py
uv run python scripts/audit_publication_readiness.py --check
uv run python scripts/z_generate_manuscript_variables.py   # run last: re-binds manuscript variables to the fi
```

The variable-injection step is run **last** on purpose: live readiness checks exercise the real project tree and may refresh benchmark-derived artifacts with newly measured wall-clock timings. Regenerating the manuscript variables afterward keeps the injected statistics and the recorded `BENCHMARK_CSV_SHA256` bound to the same CSV that ships, without treating the whole CSV hash as a cross-run reproducibility target.

Third-party containers: run `verify` before `unpack` (see sec. 14).

The reproducibility contract is deliberately operational rather than rhetorical: scripts, fixture classes, generated reports, rendered outputs, and validation commands are named so a reader can rerun or inspect the same evidence surface [Sandve et al., 2013, Wilson et al., 2017]. ENTO does not claim that this private RC already satisfies an external artifact-review badge; it does organize the release artifacts around the same availability/functionality/reusability distinctions [Association for Computing Machinery, 2024].

The project test gate requires 90% coverage on `src/` with no mocks (measured 92.29% on the latest pipeline run; no-mock scan: `Clean`). "No mocks" means real ZIP, crypto, filesystem, subprocess, and report execution over documented fixture, synthetic-stress, and conformance-vector inputs; it does not mean every input byte is a real-world observational dataset.

Benchmark CSV path: `output/data/ento_benchmark_results.csv` (SHA-256: `47f1e923ae809ab5e384e1d5d01b48f4 7de62afe04092f9922300de2a163672a`). This hash is a single-run provenance stamp, not a reproducibility target: the CSV's timing columns (`pack_seconds`, `unpack_seconds`, `pack_throughput_mib_s`) are re-measured on every run, so each regeneration yields a new CSV and a new hash. The byte-exact, run-invariant quantities are the data-derived columns — `expansion_ratio`, `manifest_bytes`, and the tamper-detection outcomes — which follow the closed-form law of eq. 4 and reproduce identically across runs.

Those deterministic columns are bound by a dedicated **data fingerprint**: `234ad77b1308c99aaaac92946a82e53687475 2e732fb99ead002f4097d0a0859`. It is the SHA-256 of only the run-invariant benchmark columns (condition, track and byte counts, expansion ratio, manifest bytes, tamper outcomes), computed row-order-independently and over canonicalized numeric values — so it depends on the *values*, not on how they happen to be spelled — by `benchmark_data_fingerprint` in `src/benchmark_stats.py`. Unlike the whole-CSV hash, this fingerprint *is* a reproducibility target — regenerating the pipeline on any host reproduces it exactly (`tests/test_data_fingerprint.py`), and a mismatch signals real corruption of the deterministic data rather than wall-clock noise. It is the honest counterpart to the closed-form expansion law: the parts of the benchmark that are determined by the format, fingerprinted; the parts that are measured, reported with their dispersion (sec. 9).

12.2 Registered figures

- `fig:benchmark_overview` – `benchmark_overview.png` (panel, results, timing-measured): Orients readers to the four headline benchmark surfaces.
- `fig:throughput_benchmark` – `throughput_benchmark.png` (scatter, results, timing-measured): Shows local pack-throughput dispersion for the medium synthetic track.
- `fig:expansion_ratio` – `expansion_ratio.png` (bar, results, data-derived): Shows per-fixture ciphertext expansion under the default profile.
- `fig:expansion_heatmap` – `expansion_heatmap.png` (heatmap, results, data-derived): Separates fixed-header overhead from payload size across benchmark conditions.
- `fig:observability_manifest_size` – `observability_manifest_size.png` (line, results, data-derived): Shows the manifest-size cost of each observability level for the EEG fixture.
- `fig:unpack_latency` – `unpack_latency.png` (bar, benchmark_interp, timing-measured): Compares local pack and verify-before-unpack wall-clock costs.
- `fig:throughput_by_observability` – `throughput_by_observability.png` (line, benchmark_interp, timing-measured): Checks whether manifest redaction changes medium-track pack throughput locally.
- `fig:observability_throughput_tradeoff` – `observability_tradeoff.png` (scatter, benchmark_interp, timing-measured): Visualizes manifest-size versus throughput coupling in the generated matrix.
- `fig:manifest_multitrack` – `manifest_multitrack.png` (line, methodology, data-derived): Compares observability

redaction behavior across heterogeneous fixture tracks.

- `fig:crypto_overhead` – `crypto_overhead.png` (bar, methodology, data-derived): Decomposes each track member into fixed AEAD header and encrypted body bytes.
- `fig:expansion_law` – `expansion_law.png` (line, benchmark_interp, data-derived): Binds measured expansion ratios to the version-aware closed-form model.
- `fig:throughput_dispersion` – `throughput_dispersion.png` (scatter, benchmark_interp, timing-measured): Shows timing variability instead of hiding it behind one mean.
- `fig:determinism_cv` – `determinism_cv.png` (bar, benchmark_interp, timing-measured): Explains why deterministic columns are fingerprinted and timing columns are not.
- `fig:format_ladder` – `format_ladder.png` (ladder, security, data-derived): Summarizes the supported wire-format ladder and default writer choice.
- `fig:format_compatibility_matrix` – `format_compatibility_matrix.png` (heatmap, security, data-derived): Makes read/write/default and hardening features explicit per format.
- `fig:length_leakage_profile` – `length_leakage_profile.png` (line, security, data-derived): Contrasts exact legacy length leakage with default PADME bucket disclosure.
- `fig:conformance_outcomes` – `conformance_outcomes.png` (heatmap, security, data-derived): Shows known-good and known-bad fixture expectations for all supported formats.
- `fig:observability_redaction_matrix` – `observability_redaction_matrix.png` (heatmap, methodology, data-derived): Shows which manifest fields survive each export level.
- `fig:release_evidence_map` – `release_evidence_map.png` (bar, reproducibility, data-derived): Maps the release candidate to its generated evidence surfaces.
- `fig:security_control_matrix` – `security_control_matrix.png` (heatmap, security, data-derived): Separates repository-enforced controls from deployment-residual controls.
- `fig:tamper_detection` – `tamper_detection.png` (bar, security, data-derived): Reports generated tag-byte tamper rejection across the benchmark matrix.

The registry lists 21 figures at 300 DPI. Each entry records `generated_by`, CSV provenance, and a `caption_token` mirrored in the manuscript figure-caption variables. After analysis, metadata is written to `../figures/figure_registry.json`.

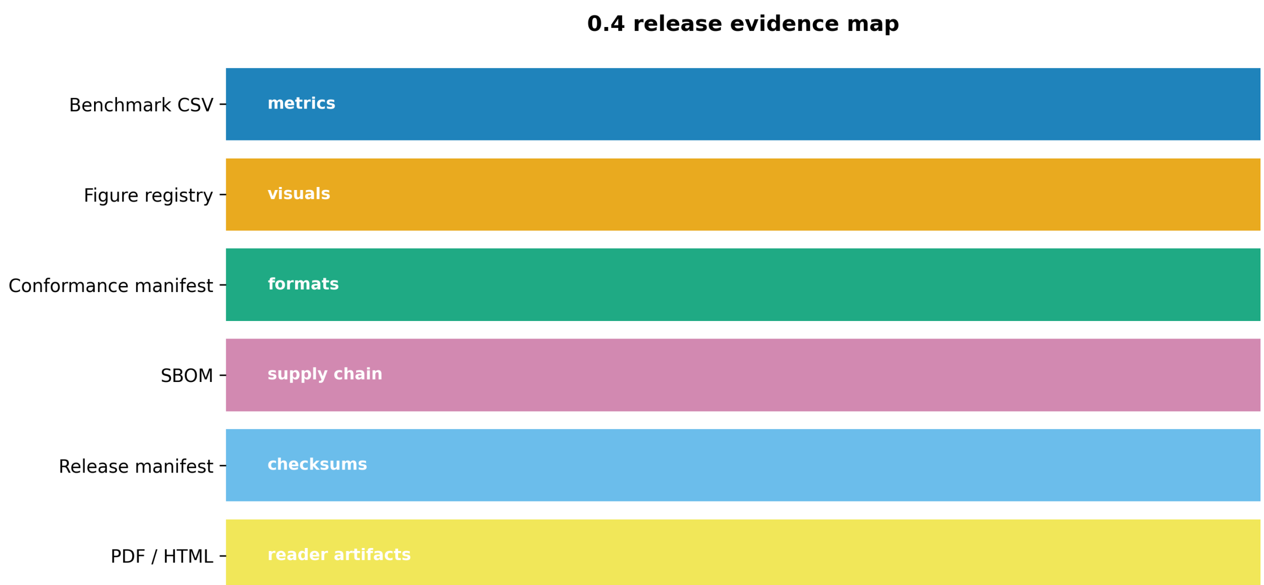

Figure 23: Release evidence map for paper 0.4 and default format 0.4.0. The figure groups generated artifacts into benchmark metrics, registered visuals, conformance vectors, SBOM, checksums, and reader-facing PDF/HTML outputs so the release candidate is auditable as an artifact set rather than prose alone. Filters: all benchmark rows. Data from `ento_benchmark_results.csv`. Generated by `generate_release_evidence_map_figure` in `src/figures.py`.

fig. 23 summarizes the generated evidence surface for paper 0.4 and default format 0.4.0: benchmark CSV, registered figures, conformance vectors, SBOM, release manifest checksums, and rendered PDF/HTML outputs.

12.3 Artifact inventory

Category	Files
Figures (`../figures/`)	benchmark_overview.png, conformance_outcomes.png, crypto_overhead.png, determinism_cv.png, expansion_heatmap.png, expansion_law.png, expansion_ratio.png, format_compatibility_matrix.png, format_ladder.png, length_leakage_profile.png, manifest_multitrack.png, observability_manifest_size.png, observability_redaction_matrix.png, observability_tradeoff.png, release_evidence_map.png, security_control_matrix.png, tamper_detection.png, throughput_benchmark.png, throughput_by_observability.png, throughput_dispersion.png, transmission_integrity_strip.png, transmission_pairing.png, unpack_latency.png
Data (`output/data/`)	_invariant.ento.zip, ento_benchmark_results.csv, manuscript_variables.json, publication_ledger.json, transmission_manifest.json
SBOM	present at `output/reports/sbom.cyclonedx.json` (components: 25)

Evidence reports: conformance status `pass` at `output/reports/conformance_report.json`; artifact manifest status `present` at `output/reports/artifact_manifest.json`; release manifest status `pass` at `output/release/release_manifest.json`. The evidence provenance contract is maintained in `docs/evidence_provenance.md`.

12.4 Fixture digests

Fixture	SHA-256
`eeg.csv`	`49a0f244623e895ec62a72579b8488cb448bc97a5294fa2d4767224d9ab385d5`
`sample.vcf`	`e78a6df9053dafc26e73f5ae5e521f68714bea7433e9aa7ab8df1e6219595e02`
`spectrogram.bin`	`fdeab9acf3710362bd2658cdc9a29e8f9c757fcf9811603a8c447cd1d9151108`

Claim bindings are declared in `data/claim_ledger.yaml` and documented in `docs/claim_ledger.md`. Tests: `tests/test_claim_ledger.py`, `tests/test_claim_ledger_security.py` (tamper rate, container verification report, GCM backend claim). The current release-candidate build path is the private `projects/working/entofile` tree rendered through the template with `--project working/entofile`; the planned public home is `https://github.com/docxology/entofile` after release readiness, not a separate source of truth for these artifacts.
Generated at: 2026-06-11T17:09:20+00:00

13 Scope and related work

ENTO intentionally scopes out streaming servers, network transport, and ledger-specific proof verification. It specifies the default on-disk layout for **0.4.0**, schema-valid supported format values (0.2.0, 0.3.0, 0.3.1, 0.4.0), and a reference Python implementation with offline verify and unpack.

13.1 Related formats

HDF5 and Zarr [hdf, 2024, zar, 2023] excel at array chunking and cloud partial reads; ENTO adds per-track authenticated encryption and ontology URIs in exchange for whole-file ZIP semantics.
EPUB and Matroska [W3C, 2023, mat, 2024] package typed media streams with container-level metadata; ENTO borrows stream typing but targets research payloads and observability redaction rather than playback.
OAIS and PREMIS [Consultative Committee for Space Data Systems, 2024, PREMIS Editorial Committee, 2015] operate above ENTO's file-format layer. OAIS describes archive responsibilities and information packages; PREMIS describes preservation metadata for objects, events, rights, and agents. ENTO is not an archival repository, a preservation-metadata data dictionary, or a certification scheme. Its release claim is narrower: a self-contained encrypted track envelope whose manifest and binary member layout are inspectable and testable.
RO-Crate, BagIt, and Frictionless Data Package [roc, 2024, Soiland-Reyes et al., 2022, Kunze et al., 2018, fri, 2024] provide preservation-, transfer-, and tabular-oriented packaging with metadata and checksum manifests. ENTO complements them: an ENTO file can sit inside a BagIt payload, be described as an RO-Crate entity, or carry exported tabular resources while keeping ciphertext semantics local to `.ento` tracks.
OpenTDF and document DRM stacks [ope, 2024] emphasize policy-bound decryption; ENTO is an offline CLI with audited GCM (`aes-256-gcm`), explicit observability levels, and reproducible open benchmarks. CADF [cad, 2013] addresses cloud audit events; ENTO proof export aligns with PROV-style derivation chains [W3C, 2013, Lebo et al., 2013] instead. External signatures and build provenance can use standard signing and supply-chain layers such as COSE, SLSA/Sigstore, or in-toto rather than changing the `.ento` ZIP internals [Schaad, 2022, SLSA, 2024, Sigstore, 2026, Torres-Arias et al., 2019]. Distilled comparisons and security norm references live in `docs/research/related_formats.md`.

13.2 Positioning

The design juxtaposition is functional: ENTO is for externally signable, typed research bundles that remain enumerable in any ZIP tool while supporting graded manifest export. The container authenticates tracks under the master key; release identity and provenance require a signature layer outside the ZIP. ENTO does not replace FAIR repository infrastructure [Wilkinson et al., 2016], OAIS archive operations [Consultative Committee for Space Data Systems, 2024], or PREMIS preservation metadata [PREMIS Editorial Committee, 2015], but supplies an encrypted track envelope those systems can catalog. Its `data/ento_track_header.ksy` file uses Kaitai Struct so the binary member layout is documented in a language-neutral parser specification [Kaitai Struct, 2026].
Frictionless Data Package [fri, 2024] targets tabular resource descriptors; ENTO track URIs serve a similar disambiguation role for heterogeneous binaries inside one archive. When migrating HDF5 [hdf, 2024] exports, treat ENTO as an integrity wrapper around extracted arrays rather than an in-place array store—chunk addressing remains the responsibility of the source format.

Layer	Primary obligation	ENTO relationship
OAIS / PREMIS [Consultative Committee for Space Data Systems, 2024, PREMIS Editorial Committee, 2015]	Archive responsibilities, preservation metadata, repository operations	External institutional and metadata layer; ENTO can be one content object inside it
RO-Crate / BagIt [roc, 2024, Soiland-Reyes et al., 2022, Kunze et al., 2018]	Research-object description, transfer packaging, checksum manifests	Complementary packaging layer; ENTO can be payload or described entity
HDF5 / Zarr [hdf, 2024, zar, 2023]	Chunked array storage and partial reads	Source or destination for plaintext arrays; ENTO does not replace chunk addressing
OpenTDF [ope, 2024]	Policy-bound decryption	Adjacent policy model; ENTO is offline and key-file based
ENTO 0.4.0	Per-track AEAD, typed manifest, observability redaction	Implemented file-format layer validated by this release candidate

See sec. 14 for threat-model boundaries and sec. 9 for measured trade-offs between observability levels and manifest size.

14 Limitations and threat model

ENTO is a reference container for reproducible research bundles—not a full data-management platform or enterprise DRM system. This section states explicit limits and the adversary model the implementation addresses.

14.1 Scope limits

- **No streaming partial decrypt**: tracks unpack as whole ZIP entries; HDF5/Zarr-style chunk random access is out of scope [hdf, 2024, zar, 2023].
- **No online policy engine**: unlike OpenTDF [ope, 2024], decryption does not consult remote attribute authorities.
- **ZIP metadata leakage**: file names (`tracks/eeg.ento`) and compressed sizes remain visible at every level. Observability redaction applies only to per-*track* manifest fields (type, digests, resolution, `byte_length`), and not uniformly: `byte_length` is redacted (zeroed) **only at the sealed level**, whereas type/digests are redacted at lower levels too — so a non-sealed manifest still publishes `byte_length`. The manifest *header* — `creator`, `created` timestamp, and `format_version` — is carried in cleartext even at the sealed level, so it must not hold sensitive content (strip or generalize `creator`/`created` before sealed distribution if they are sensitive).
- **Plaintext length**: AES-GCM is length-preserving for the bytes supplied to encryption, so unpadded compatibility formats reveal exact plaintext length from member size. Default 0.4.0 PADMÉ-pads the encrypted body, so the on-disk ciphertext member size reveals only a coarse bucket. This padding addresses the *member-size* channel only: at non-sealed observability levels (including the default AUDITABLE) the manifest still carries the per-track `byte_length` field in cleartext, which discloses exact plaintext length directly — fully nullifying PADMÉ for those containers. No export level hides length exactly. Sealed export redacts the `byte_length` field to zero, but the on-disk member size still reveals the PADMÉ *bucket* (coarse for large payloads, near-exact for small ones, e.g. distinct small lengths can fall in distinct buckets); and for unpadded compatibility formats the member size reveals exact length at every level. The strongest available length-hiding is therefore sealed export of a padded format, and even then only to bucket granularity.
- **Archival-system boundary**: OAIS repository responsibilities and PREMIS preservation metadata are external to the ENTO file format [Consultative Committee for Space Data Systems, 2024, PREMIS Editorial Committee, 2015]. ENTO can be stored in those systems, but this repository does not implement appraisal, accession policy, preservation planning, rights management, repository certification, or long-term media migration.
- **Evidence provenance**: release outputs are real executions of the repository pipeline, but benchmark and conformance inputs include deterministic fixtures, a synthetic throughput stress track, and deterministic test vectors. See `docs/evidence_provenance.md`.
- **Format ladder**: 0.4.0 is the default on-disk contract. Compatibility formats 0.2.0, 0.3.0 and 0.3.1 are version-dispatched and remain readable/writable alongside it; older experimental ciphertext layouts are out of scope for this release.

14.2 Threat model

Adversary capability	Mitigation	Residual risk
Tamper ciphertext or tags	AES-256-GCM authenticates on key-based unpack/verify (the integrity anchor) [National Institute of Standards and Technology, 2001, Dworkin, 2007, McGrew and Viega, 2004]	Keyless digest check detects accidental corruption only
Swap manifest after export	Key-based decrypt mismatch; `verify_proof_export` consistency check [Rundgren et al., 2020, Merkle, 1988, Haber and Stornetta, 1991]	Unkeyed proof is forgeable; sign externally for origin
Misreport verification assurance	`verify` reports `key-authenticated`/`digest-only`/`unverified` and fails closed by default	`--allow-unverified` opt-out is the operator's risk
Escape output path via track id	IDs `[a-z0-9._-]+`; `safe_output_path`	Path traversal remains the class being excluded [MITRE Corporation, 2024a]
ZIP bomb / extra members	Actual-byte read bound + aggregate budget; member-set equality; duplicate-name rejection	Data-amplification limits are tunable per deployment [MITRE Corporation, 2024b]

Adversary capability	Mitigation	Residual risk
Infer plaintext length from track size	Sealed export zeroes manifest `byte_length`; **0.4.0 PADMÉ-pads** the ciphertext member	Bucket size remains visible; the cleartext manifest `byte_length` reveals exact length at every non-sealed level (so PADMÉ length-hiding applies only under sealed export); unpadded compatibility formats also reveal exact length from member size
Downgrade hardened format	0.4.0 and AAD-bound compatibility formats bind `format_version` in associated data	Legacy no-AAD compatibility relies on track-key binding and external policy
Guess master key	256-bit key; per-track HKDF separation	Key storage outside ENTO scope
Learn types at sealed export	Level 0 strips type URIs and hashes	Track filenames may hint content
Replay old manifest	Not addressed	External timestamping

Full AppSec table and MITRE ATT&CK mapping: `docs/entofile-threat-model.md`. Operational checklist: sec. 6. For the planned public repository, the same boundary applies: GitHub release signatures, SBOM attestations, KMS/HSM key custody, and SOC routing can strengthen deployment assurance, but they are not properties of an `.ento.zip` file unless the operator adds those controls around it. NIST key-management guidance and supply-chain-risk guidance are therefore cited as deployment requirements, not as claims that this offline reference repository already implements a vault, release attestation service, or provenance control plane [National Institute of Standards and Technology, 2020, 2022].

14.3 Security hardening and format evolution

The reference implementation layers three honesty- and hardening-oriented guarantees on the 0.4.0 default:

1. **Honest verification.** Keyless verification is corruption-detection only; the manifest digests and hash-chained proof are unkeyed and forgeable by anyone who controls the bytes. Adversarial integrity comes solely from AES-256-GCM authentication under the master key. `verify` surfaces this distinction in its `integrity` field and fails closed on unverifiable input (sec. 6).
2. **Format 0.4.0** uses the SP 800-38D-standard 12-byte nonce and binds `format_version` and the track id as AEAD associated data [McGrew, 2008, Dworkin, 2007], so a format downgrade or cross-track relabel fails authentication. This is also why nonce uniqueness is a hard invariant rather than a performance detail: repeated GCM nonces enable practical forgery and confidentiality failures [Joux, 2006, Böck et al., 2016].
3. **Default PADMÉ padding** [Nikitin et al., 2019] length-prefixes each plaintext and pads it to a coarse bucket before encryption, so the on-disk ciphertext size reveals only the bucket (overhead O(log log L)), not the exact length. This mitigates the plaintext-length side-channel for deployments where length analysis is in scope; it hides length to bucket granularity, not perfectly. Padding addresses the ciphertext member-size channel only, and only to bucket granularity (never exact): the manifest `byte_length` field still publishes exact length at non-sealed observability levels, and even sealed export leaves the padded member size revealing the bucket. The strongest length-hiding is sealed export of a padded format, bounded to bucket granularity.

If an operator cannot rely on fresh random nonces per per-track key, a nonce-misuse-resistant AEAD such as AES-GCM-SIV is a future-format candidate, not an ENTO 0.4.0 behavior [Gueron and Lindell, 2019].

14.4 Nation-state pillar status

Pillar	Status
Verify-before-use (ZTA)	`verify` CLI + `container_verification.json` gate, aligned with zero-trust verification principles [Rose et al., 2020]
Cryptography	GCM default 0.4.0 plus compatibility formats 0.2.0, 0.3.0 and 0.3.1; PQC standards inform external transport/signing rather than this symmetric file envelope [National Institute of Standards and Technology, 2001, 2015, Dworkin, 2007, National Institute of Standards and Technology, 2024b,a]

Pillar	Status
Supply chain	Optional CycloneDX SBOM (`scripts/export_sbom.py`); external signing/provenance should use NIST C-SCRM, SLSA/Sigstore, COSE, or in-toto-style release controls [OWASP CycloneDX, 2026, National Institute of Standards and Technology, 2022, SLSA, 2024, Sigstore, 2026, Schaad, 2022, Torres-Arias et al., 2019]
Detection	Structured verify failure JSON; deployment SOC mapping can use MITRE ATT&CK review coverage [MITRE, 2026]

See `docs/nation_state_roadmap.md` for HSM, signing, and audit integrations.

14.5 Key handling

Master keys are 32 random bytes from `genkey`. The CLI sets mode `0600` on Unix when writing key files. Run `verify` before `unpack` on third-party archives. Escrow, cryptoperiods, access control, and HSM/KMS policies are deployment concerns governed outside ENTO's file format [National Institute of Standards and Technology, 2020] (`docs/security.md`).

14.6 Non-goals

CADF audit streams [cad, 2013], FAIR repository automation [Wilkinson et al., 2016], OAIS/PREMIS preservation operations [Consultative Committee for Space Data Systems, 2024, PREMIS Editorial Committee, 2015], and RO-Crate aggregation [roc, 2024, Soiland-Reyes et al., 2022] are complementary–ENTO specifies the encrypted track envelope they might wrap. Future work may add chunked tracks, external KMS hooks, and formal interoperability tests without changing the 0.4.0 on-disk contract.

15 References

Bibliography lives in `manuscript/references.bib` and is read by Pandoc during PDF render. Inline citations use `[@citekey]` syntax throughout introduction, methodology, related work, and limitations sections.

Validate bibliography syntax:

```
uv run python -m infrastructure.reference.citation.cli validate \
    projects/working/entofile/manuscript/references.bib --strict
```

Research notes backing related-work claims: `../docs/research/related_formats.md`.

References

Cloud audit data federation (CADF). https://www.dmtf.org/standards/cadf, 2013.

JSON Schema: A media type for describing JSON documents. https://json-schema.org/draft/2020-12/json-schema-core.html, 2020.

Zarr specification. https://zarr-specs.readthedocs.io/, 2023.

Frictionless data package specification. https://specs.frictionlessdata.io/data-package/, 2024.

HDF5 file format reference. https://support.hdfgroup.org/documentation/hdf5/latest/, 2024.

Matroska element specification. https://www.matroska.org/technical/specs/index.html, 2024.

OpenTDF specification. https://opentdf.io/, 2024.

RO-Crate metadata specification. https://www.researchobject.org/ro-crate/, 2024.

APPNOTE.TXT — ZIP file format specification. https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT, 2024.

Association for Computing Machinery. Artifact review and badging. ACM Publications Policy, 2024. URL https://www.acm.org/publications/policies/artifact-review-and-badging-current.

Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. Nonce-disrespecting adversaries: Practical forgery attacks on GCM in TLS. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*. USENIX Association, 2016. URL https://www.usenix.org/conference/woot16/workshop-program/presentation/bock.

Consultative Committee for Space Data Systems. Reference model for an open archival information system (OAIS). Technical Report CCSDS 650.0-M-3, Consultative Committee for Space Data Systems, 2024. URL https://public.ccsds.org/Pubs/650x0m3.pdf.

Morris Dworkin. Recommendation for block cipher modes of operation: Galois/Counter mode (GCM) and GMAC. Technical Report SP 800-38D, National Institute of Standards and Technology, 2007. URL https://csrc.nist.gov/pubs/sp/800/38/d/final.

Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering*. Wiley, 2010.

Shay Gueron and Yehuda Lindell. AES-GCM-SIV: Nonce misuse-resistant authenticated encryption. RFC 8452, IRTF, 2019. URL https://www.rfc-editor.org/info/rfc8452.

Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991. doi: 10.1007/BF00196791. URL https://link.springer.com/article/10.1007/BF00196791.

Antoine Joux. Authentication failures in NIST version of GCM. Comments submitted to the NIST Modes of Operation process, 2006. URL https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/800-38-series-drafts/gcm/joux_comments.pdf.

Kaitai Struct. Kaitai struct: Declarative binary format parsing language, 2026. URL https://kaitai.io/.

Hugo Krawczyk and Pasi Eronen. Hmac-based extract-and-expand key derivation function (HKDF), 2010. URL https://www.rfc-editor.org/info/rfc5869.

John Kunze, Justin Littman, Liz Madden, John Scancella, and Chris Adams. The BagIt file packaging format (RFC 8493). RFC 8493, RFC Editor, 2018. URL https://www.rfc-editor.org/info/rfc8493.

Timothy Lebo, Satya Sahoo, and Deborah McGuinness. PROV-O: The PROV ontology. W3C Recommendation, 2013. URL https://www.w3.org/TR/prov-o/.

David McGrew. An interface and algorithms for authenticated encryption. RFC 5116, IETF, 2008. URL https://www.rfc-editor.org/info/rfc5116.

David A. McGrew and John Viega. The security and performance of the Galois/Counter mode (GCM) of operation. In *Progress in Cryptology – INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004. doi: 10.1007/978-3-540-30556-9_27. URL https://doi.org/10.1007/978-3-540-30556-9_27.

Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology – CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1988. doi: 10.1007/3-540-48184-2_32. URL https://link.springer.com/chapter/10.1007/3-540-48184-2_32.

MITRE. MITRE ATT&CK. https://attack.mitre.org/, 2026.

MITRE Corporation. CWE-22: Improper limitation of a pathname to a restricted directory (path traversal). Common Weakness Enumeration, 2024a. URL https://cwe.mitre.org/data/definitions/22.html.

MITRE Corporation. CWE-409: Improper handling of highly compressed data (data amplification). Common Weakness Enumeration, 2024b. URL https://cwe.mitre.org/data/definitions/409.html.

National Institute of Standards and Technology. Advanced encryption standard (AES). Technical Report FIPS PUB 197, U.S. Department of Commerce, 2001. URL https://csrc.nist.gov/pubs/fips/197/final.

National Institute of Standards and Technology. Secure hash standard (SHS). Technical Report FIPS PUB 180-4, U.S. Department of Commerce, 2015. URL https://csrc.nist.gov/pubs/fips/180-4/upd1/final.

National Institute of Standards and Technology. Recommendation for key management: Part 1 – general. Technical Report SP 800-57 Part 1 Revision 5, National Institute of Standards and Technology, 2020. URL https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final.

National Institute of Standards and Technology. Cybersecurity supply chain risk management practices for systems and organizations. Technical Report SP 800-161 Revision 1 Update 1, National Institute of Standards and Technology, 2022. URL https://csrc.nist.gov/pubs/sp/800/161/r1/upd1/final.

National Institute of Standards and Technology. Module-lattice-based digital signature standard. Technical Report FIPS 204, U.S. Department of Commerce, 2024a. URL https://csrc.nist.gov/pubs/fips/204/final.

National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Technical Report FIPS 203, U.S. Department of Commerce, 2024b. URL https://csrc.nist.gov/pubs/fips/203/final.

Kirill Nikitin, Ludovic Barman, Wouter Lueks, Matthew Underwood, Jean-Pierre Hubaux, and Bryan Ford. Reducing metadata leakage from encrypted files and communication with PURBs. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2019(4):6–33, 2019. doi: 10.2478/popets-2019-0056. URL https://petsymposium.org/popets/2019/popets-2019-0056.php.

NIST. Secure software development framework (SSDF) version 1.1. Technical Report SP 800-218, NIST, 2022.

OWASP CycloneDX. CycloneDX specification. https://github.com/CycloneDX/specification, 2026.

PREMIS Editorial Committee. PREMIS data dictionary for preservation metadata, version 3.0. Library of Congress Standards, 2015. URL https://www.loc.gov/standards/premis/v3/index.html.

Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. Technical Report SP 800-207, NIST, 2020.

Anders Rundgren, Bret Jordan, and Samuel Erdtman. JSON canonicalization scheme (JCS). RFC 8785, IETF, 2020. URL https://www.rfc-editor.org/info/rfc8785.

Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten simple rules for reproducible computational research. *PLOS Computational Biology*, 9(10):e1003285, 2013. doi: 10.1371/journal.pcbi.1003285. URL https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003285.

Jim Schaad. CBOR object signing and encryption (COSE): Structures and process. RFC 9052, IETF, 2022. URL https://www.rfc-editor.org/info/rfc9052.

Sigstore. Sigstore Cosign signing documentation. https://docs.sigstore.dev/cosign/signing/, 2026.

SLSA. SLSA version 1.0 security levels. https://slsa.dev/spec/v1.0/levels, 2024.

Stian Soiland-Reyes, Peter Sefton, Mercè Crosas, Leyla Jael Castro, Frederik Coppens, José M. Fernández, Daniel Garijo, Björn Grüning, Marco La Rosa, Simone Leo, Eoghan Ó Carragáin, Marc Portier, Ana Trisovic, Paul Groth, and Carole Goble. Packaging research artefacts with RO-Crate. *Data Science*, 5(2):97–138, 2022. doi: 10.3233/DS-210053. URL https://doi.org/10.3233/DS-210053.

Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *28th USENIX Security Symposium*, pages 1393–1410. USENIX Association, 2019. URL https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias.

W3C. PROV-DM: The PROV data model. https://www.w3.org/TR/prov-dm/, 2013.

W3C. EPUB 3.3. https://www.w3.org/TR/epub-33/, 2023.

Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3:160018, 2016. doi: 10.1038/sdata.2016.18. URL https://www.nature.com/articles/sdata201618.

Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. Good enough practices in scientific computing. *PLOS Computational Biology*, 13(6):e1005510, 2017. doi: 10.1371/journal.pcbi.1005510. URL https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510.

END OF TRANSMISSION

Release: v0.4 · DOI `10.5281/zenodo.20396329` · SHA-256 `05bcfb08323a…` · pairing complete

Metadata Citation Contact Integrity Code128

Zenodo GitHub Manifest

T:ENTO: an ENcrypted, Typed, Omnitra

Figure 24: Integrity QR strip

Prior: *No prior releases.*