Independent Submission
Internet-Draft
Intended status: Informational
Expires: 6 November 2026

T. En Nakhai
Crovia Trust

May 2026

The Crovia Seal: A Cryptographic Receipt Format for AI-Generated Output Provenance

draft-crovia-seal-01

Abstract

This document specifies the Crovia Seal v1, a compact, tamper-evident JSON receipt that may be attached to any output produced by an AI generator (large language model, image model, audio model, or composite system) to record its provenance in a cryptographically verifiable form. A Crovia Seal binds an issuer’s identity to the SHA-256 digests of an input/output pair, the identity and parameters of the generator, an emission timestamp, and a per-issuer hash chain, under an Ed25519 signature computed over a strict canonicalization (CSC-1) of the receipt with explicit domain separation. Optional fields permit transparency-log inclusion proofs and witness co-signatures. The Seal is designed for offline verification and for inclusion in third-party transparency logs and standards-based revocation infrastructure.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

This Internet-Draft will expire on 5 November 2026.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Conventions and Terminology 4
  1.2. Design Principles 4
  1.3. Out of Scope 5
2. Canonicalization (CSC-1) 5
  2.1. Motivation 5
  2.2. Rules 6
  2.3. Signing Payload 6
3. Seal Structure 7
  3.1. Top-Level Fields 7
  3.2. seal_version 8
  3.3. seal_id 8
  3.4. issuer 8
  3.5. subject 9
  3.6. generator 9
  3.7. timestamp 9
  3.8. chain 9
  3.9. checks (OPTIONAL) 10
  3.10. anchor (OPTIONAL) 10
  3.11. signature 11
4. Witnesses (OPTIONAL) 11
5. Transparency Log (Informative) 11
6. Algorithm Agility (Post-Quantum) 12

7. Security Considerations 12
7.1. Replay Across Protocols 12
7.2. Replay Within the Protocol 12
7.3. JSON Malleability 12
7.4. Field Tampering 12
7.5. Key Compromise 12
7.6. Downgrade 13
7.7. Canonicalization Ambiguity 13
7.8. Hash Choice 13
7.9. Randomness 13
7.10. Content Privacy 13
8. IANA Considerations 13
8.1. Media Type Registration 13
8.2. URN Sub-Namespace Registration 14
9. References 14
9.1. Normative References 14
9.2. Informative References 15
Appendix A. Reference Implementation 15
Appendix B. Changelog 15
Acknowledgements 16
Author’s Address 16

1. Introduction

Outputs produced by automated generative systems (large language models, diffusion image models, speech-synthesis pipelines, or composite agents) are difficult to attribute reliably after they leave the generating system. Today, an output may be quoted, edited, copy-pasted, ingested into a search index, surfaced in a court filing, or attributed to the wrong vendor with no available chain of custody. Both vendors and downstream consumers have an operational interest in an attestation that fixes, at the moment of generation, the bytes that were produced and the parameters under which they were produced.

This document specifies a single such attestation: the *Crovia Seal*. A Seal is a JSON object whose bytes are signed by an issuer’s Ed25519 private key after a deterministic canonicalization (Section 3) and with explicit cryptographic domain separation (Section 3.3). A Seal carries the SHA-256 digest of the input that was provided to the generator, the SHA-256 digest of the output, the generator identity and generation parameters, an issuance timestamp, and a per-issuer append-only hash chain. Optional fields carry inclusion proofs from public transparency logs and additional co-signatures ("witnesses").

A Seal does *not* attest to the truthfulness, lawfulness, originality, or safety of the output. It is a receipt, not a verdict. This deliberate narrowness is essential to the protocol’s

value: the Seal can be re-verified by any party possessing only the Seal, the issuer's public key, and the original output bytes, without dependence on any judgement that cannot be cryptographically checked.

1.1. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are used throughout this document:

Seal: a single JSON object conforming to Section 3.

Issuer: the entity producing and signing Seals; identified by its long-term Ed25519 public key.

Witness: an optional co-signer of a Seal (Section 4).

Subject: the input/output pair that the Seal describes.

Generator: the AI model or system that produced the output.

Chain: the per-issuer append-only sequence of Seals (Section 3.8).

Anchor: an optional proof that the Seal was included in a public transparency log (Section 5).

CSC-1: Crovia Seal Canonicalization v1 (Section 2).

Payload: the exact byte sequence over which a signature is computed (Section 2.3).

1.2. Design Principles

* *Record, do not judge.* The Seal MUST NOT encode verdicts on lawfulness, harm, or quality. It MAY embed analytical signals with explicit confidence.

* *Verify offline.* The core verification path MUST require no network access.

* *Composable.* The schema MUST permit optional extensions (co-signatures, post-quantum signatures, transparency anchors) without breaking base verifiers.

* *Canonical.* The exact bytes that are signed MUST be derivable from the Seal by a single, fully specified canonicalization algorithm.

* *Defensive by default.* All cryptographic operations MUST use domain separation; no signed payload may be replayed in another protocol.

* *Fail closed.* Any unrecognized field or algorithm MUST cause verification to fail unless the verifier explicitly opts into permissive mode.

1.3. Out of Scope

The following are explicitly out of scope for this specification. An attestation that silently included any of them would be misunderstood by users:

* Detection of AI-generated content from the output alone (no watermarking).

* Judgements about copyright, fairness, accuracy, or safety of the output.

* Revocation: a Seal, once emitted, is an immutable historical record. Issuers are identified by public key; key rotation is handled by the transparency log, not by Seal-level revocation.

* Authentication of the human user that requested the generation. The Seal binds bytes, not personhoods.

* Confidentiality. Seals are over plaintext.

2. Canonicalization (CSC-1)

2.1. Motivation

JSON [RFC8259] is syntactically flexible: whitespace, key ordering, number formatting, and string-escape choices can vary while producing the same logical value. A signature over a JSON text requires an unambiguous serialization.

CSC-1 is a strict subset of the JSON Canonicalization Scheme (JCS) [RFC8785]. It adopts the deterministic ordering and escaping rules of JCS but forbids floating-point numbers in signed payloads, sidestepping the edge cases of ECMA-262 number serialization. This restriction does not constrain use cases: continuous parameters such as temperature MUST be encoded as strings when carried inside the signed payload (Section 3.6).

2.2. Rules

A CSC-1-serialized JSON value is a UTF-8 byte sequence produced as follows:

1. null -> null

2. true / false -> true / false

3. integers (JSON number with no fractional part, within [-(2^53 - 1), 2^53 - 1]) -> shortest decimal representation, no leading zeros, no + sign.

4. strings -> UTF-8 JSON string literal using only the escapes required by [RFC8259]: \", \\, \b, \f, \n, \r, \t, and \u00XX for U+0000..U+001F. All other code points MUST be emitted literally.

5. arrays -> [followed by canonicalized elements separated by , followed by]. No interior whitespace.

6. objects -> { followed by "key":value pairs separated by , followed by }. Keys MUST be sorted ascending by their UTF-16 code-unit sequence (equivalent to JavaScript Array.prototype.sort on strings). No interior whitespace.

7. floats, NaN, Infinity, and -0 MUST cause serialization to fail with NonCanonicalNumber.

8. duplicate object keys MUST cause serialization to fail with DuplicateKey.

9. non-string object keys MUST cause serialization to fail with NonStringKey.

2.3. Signing Payload

Given a Seal _S_, the signing payload _P(S)_ is computed as:

P(S) = DOMAIN || 0x0A || CSC1(S \ {signature, witnesses})

where:

* DOMAIN is the ASCII string "CROVIA-SEAL-v1" (14 bytes).

* 0x0A is a single newline byte, acting as an unambiguous separator.

* S \ {signature, witnesses} denotes the Seal with the signature and witnesses top-level fields removed (they are computed over the payload, not part of it).

* CSC1(...) is the UTF-8 serialization per Section 2.2.

The DOMAIN prefix ensures that a signature over _P(S)_ cannot be replayed as a valid signature in any other protocol that does not use the same prefix. Implementations MUST NOT omit the prefix. Verifiers MUST reject any Seal whose signature was produced without the prefix.

3. Seal Structure

3.1. Top-Level Fields

A conformant Seal is a JSON object with exactly the top-level fields listed below. Unknown top-level fields MUST cause verification to fail (the "fail closed" principle).

Field	Required	Type	Section
seal_version	MUST	string	4.2
seal_id	MUST	string	4.3
issuer	MUST	object	4.4
subject	MUST	object	4.5
generator	MUST	object	4.6
timestamp	MUST	object	4.7
chain	MUST	object	4.8
checks	OPTIONAL	object	4.9
anchor	OPTIONAL	object	4.10
signature	MUST	object	4.11
witnesses	OPTIONAL	array	6

Table 1

3.2. seal_version

The literal string "crovia.seal.v1". Any other value MUST cause verification to fail.

3.3. seal_id

A string matching the regular expression ^cs_[0-9]{4}_[A-Z2-7]{26}$: prefix cs_, 4-digit issuance year, underscore, 26 RFC 4648 base32 characters [RFC4648] (alphabet A-Z, 2-7, no padding) encoding 16 random bytes (128 bits). The random bytes MUST be produced by a cryptographically secure source.

3.4. issuer

```
{
  "id":     string,           ; urn:crovia:seal-issuer:<name>
  "pubkey": { "alg": "ed25519", "key_hex": string }
}
```

key_hex is 64 lowercase hexadecimal characters (32 raw bytes, Ed25519 public key per [RFC8032]). Other algorithms are reserved for Section 6.

3.5. subject

```
{
  "input_hash":  "sha256:" + 64 lowercase hex chars,
  "output_hash": "sha256:" + 64 lowercase hex chars,
  "input_len":   integer,    ; byte length of input
  "output_len":  integer,    ; byte length of output
  "modality":    string      ; one of: text/code/image/audio/multimodal
}
```

The Seal does NOT carry the content itself. The hashes commit to the content; verifiers who possess the content can re-hash and compare.

3.6. generator

```
{
  "id":           string,          ; e.g. "openai/gpt-4o"
  "version":      string | null,   ; e.g. "2024-08-06"
  "weights_hash": string | null,   ; if available
  "params":       object           ; key -> string map of gen-params
}
```

All parameter values MUST be strings in the signed payload (per the floating-point restriction in Section 2). Numeric values like temperature=0.7 MUST be encoded as "0.7".

3.7. timestamp

```
{
  "emitted_at": string,   ; RFC 3339 UTC, ms precision
                          ; e.g. "2026-04-15T12:34:56.789Z"
  "nonce":      string    ; 26 RFC 4648 base32 chars (16 random bytes)
}
```

emitted_at follows [RFC3339]. Issuers MUST use UTC. The nonce guarantees that two Seals issued in the same millisecond are still distinct under P(S).

3.8. chain

```
{
  "prev_seal_hash": "sha256:" + 64 hex chars | null,
  "sequence":       integer (>= 0)
}
```

prev_seal_hash is the SHA-256 over the canonical payload _P(S_prev)_ of the immediately preceding Seal from the same issuer, or null for the genesis Seal (sequence == 0). Verifiers that track issuer chains MUST detect:

* *Fork*: two Seals with the same issuer key and same chain.sequence but different prev_seal_hash. This is non-repudiable evidence of issuer misbehavior or key compromise.

* *Gap*: missing sequence numbers. A verifier with partial history SHOULD obtain the missing Seals from the transparency log before accepting.

3.9. checks (OPTIONAL)

Free-form object carrying analytical claims such as memorization checks, safety probes, toxicity scores. The schema for specific check types is defined in separate check specifications; the Seal itself imposes no constraints on the content beyond CSC-1 compatibility.

Example:

```
{
  "memorization": {
    "db_version": "crovia-memdb-2026-04-15",
    "method":     "ngram-lsh-v1",
    "matches":    0,
    "max_conf":   "0.03"
  }
}
```

Each check produced by an issuer is signed along with the rest of the Seal; its validity as evidence depends on the method’s own robustness, which is out of scope for this specification.

3.10. anchor (OPTIONAL)

```
{
  "log_url":        string,
  "merkle_root":    "sha256:" + 64 hex chars,
  "merkle_proof":   [ "sha256:" + hex, ... ],
  "log_index":      integer (>= 0),
  "root_signed_at": string (RFC 3339 UTC)
}
```

The anchor commits the Seal to a public transparency log. The log operator's signature over merkle_root is not part of this Seal and MUST be fetched separately from log_url.

3.11. signature

```
{
  "alg":              "ed25519",
  "canon":            "csc-1",
  "domain":           "CROVIA-SEAL-v1",
  "payload_hash_alg": "sha256",
  "sig_hex":          string     ; 128 hex chars (64 raw bytes)
}
```

The signature is computed as Ed25519_sign(privkey, P(S)) where _P(S)_ is defined in Section 2.3. Implementations MUST NOT hash the payload before signing; Ed25519 internally handles the hash (SHA-512). The payload_hash_alg field is informational, indicating the algorithm used to derive prev_seal_hash and other SHA-256 digests in the Seal.

4. Witnesses (OPTIONAL)

```
"witnesses": [
  {
    "id":      string,
    "pubkey":  { "alg": "ed25519", "key_hex": string },
    "sig_hex": string
  },
  ...
]
```

A witness signs the same canonical payload _P(S)_ as the issuer. Witness signatures are OPTIONAL and additive: a Seal with no witnesses is valid; a Seal with invalid witness signatures is invalid overall (fail-closed).

Typical witnesses: consortium co-signers (civil-society organizations, academic institutions, regulatory observers). Witnesses endorse the Seal without endorsing its content.

5. Transparency Log (Informative)

A conformant transparency log accepts Seals and periodically publishes a signed Merkle root over the Seals it has received. The log API design is outside the scope of this document; an implementation profile compatible with [RFC9162]-style logs is anticipated in a companion document.

6. Algorithm Agility (Post-Quantum)

Future versions of this specification MAY permit alternative signature algorithms (e.g., Dilithium, Falcon) by extending the signature.alg vocabulary. A Seal MAY additionally carry a pq_signature top-level field with an independent post-quantum signature over the same payload _P(S)_. Verifiers that support only Ed25519 MUST ignore pq_signature and rely on signature. Verifiers MAY require both signatures to validate (strict mode).

7. Security Considerations

7.1. Replay Across Protocols

Prevented by domain separation (Section 2.3). A signature on _P(S)_ cannot be reinterpreted as a valid signature on a payload of any other protocol that does not use the exact same 14-byte CROVIA-SEAL-v1 prefix followed by a newline.

7.2. Replay Within the Protocol

A Seal is a historical record; "replay" of a Seal is a semantic issue, not a cryptographic one. The seal_id, timestamp, and chain.sequence fields make each Seal unique. Verifiers that track seen seal IDs can detect duplication attempts.

7.3. JSON Malleability

Eliminated by CSC-1 (Section 2). Any tool that reorders keys, adds whitespace, or re-escapes strings produces a different byte sequence and hence an invalid signature.

7.4. Field Tampering

The signature covers every field except signature and witnesses, which are themselves cryptographic. Adding, modifying, or deleting any field invalidates the signature.

7.5. Key Compromise

If an issuer key is compromised, the attacker can issue valid Seals until the compromise is detected and the key is revoked in the transparency log’s trust root. The issuer hash chain MAY reveal the compromise if the attacker issues a forking Seal.

7.6. Downgrade

seal_version, signature.alg, signature.canon, and signature.domain are all inside the signed payload. An attacker cannot negotiate a weaker algorithm without producing a wholly new signature.

7.7. Canonicalization Ambiguity

CSC-1 forbids floats in the signed payload precisely to avoid the numeric edge cases of JCS. Strings carrying numeric values MUST use a documented format (see Section 3.6).

7.8. Hash Choice

SHA-256 is used for all digests (content commitments, chain links). Migration to SHA-3 or BLAKE3 is anticipated in a future minor version; the payload_hash_alg field signals the choice.

7.9. Randomness

All random values (seal_id suffix, timestamp.nonce) MUST be produced by a cryptographically secure source (e.g., os.urandom, the secrets module in Python, crypto.randomBytes in Node.js).

7.10. Content Privacy

The Seal commits to input/output hashes, never content. Content-bearing fields such as generator.params are in the clear; issuers MUST NOT place sensitive user data in these fields.

8. IANA Considerations

8.1. Media Type Registration

IANA is requested to register the following media type per [RFC6838]:

Type name: application

Subtype name: vnd.crovia.seal+json

Required parameters: none

Optional parameters: none

Encoding: UTF-8 (per [RFC8259])

Security considerations: see Section 7

Interoperability: see this document

Published specification: this document

Applications using this type: provenance attestation systems for AI-generated content

Author/Change controller: CROVIA Research

8.2. URN Sub-Namespace Registration

IANA is requested to register the URN namespace urn:crovia:seal-issuer: per [RFC8141] for the identification of Crovia Seal issuers by stable URN. The complete formal namespace registration template will be provided in a companion document.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, June 2020, <https://www.rfc-editor.org/info/rfc8785>.

9.2. Informative References

[C2PA] Coalition for Content Provenance and Authenticity, "C2PA Content Credentials Specification, Version 1.3", 2024, <https://c2pa.org/specifications/specifications/1.3/specs/C2PA_Specification.html>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, December 2021, <https://www.rfc-editor.org/info/rfc9162>.

[SIGSTORE] The Linux Foundation, "Sigstore: Software signing for everyone", 2021, <https://www.sigstore.dev/>.

Appendix A. Reference Implementation

A reference implementation in Python and TypeScript, including a public sealing service, an in-browser verifier, and a set of normative test vectors, is available at https://croviatrust.com/registry/seal/. Test vectors are linked from https://croviatrust.com/registry/seal/spec/.

Appendix B. Changelog

draft-crovia-seal-01 (May 2026):

* Editorial: replaced 6 Unicode arrow characters (U+2192) with ASCII -> so the rendered text file is strictly ASCII-clean (0 idnits warnings). No normative changes.

draft-crovia-seal-00 (May 2026):

* Initial submission. Aligned with Crovia Seal Specification v0.5 of April 2026.

Acknowledgements

The Crovia Seal has been informed by extensive prior work in cryptographic transparency ([RFC9162], [SIGSTORE]) and media-provenance attestation ([C2PA]). The authors thank the broader community working on AI provenance and tamper-evidence for the intellectual foundations on which this work rests.

Author’s Address

Tarik En Nakhai
Crovia Trust
Italy
Email: info@croviatrust.com
URI: https://croviatrust.com