Machine Limits Standard

Draft 0.1 — a format for machine records and the limits the machine declares about them

Publisher: Machine Accountability Corporation
Published at: machineaccountability.org/standard
Status: Draft 0.1, November 2026. This draft may change without notice.
Patent notice: Patents or patent applications may cover implementations of this standard. An implementation license is available from Machine Accountability Corporation. Free licenses are available for non-commercial, research, evaluation and conformance-testing use.

1. What this standard is for

A machine or software agent that acts in the world produces records of what it sensed and did. Those records are only as useful as the reasons a reader has to trust them. This standard defines a format, called an **evidence package**, that lets a reader check, without contacting the machine's operator:

1. that the released records are the records the machine sealed;
2. the conditions that the machine itself reported as making its records less reliable during the period, and, for each type of condition, whether the machine monitored for it;
3. which maker key attested the machine's key, and the software and model measurements the machine stated;
4. the legal process that the custodian states as the basis for release;
5. who certified the package, and whether that certifier was listed in a registry at the time; and
6. where several machines recorded the same facts, the compiler's statement of which agree, which disagree, and which did not observe.

A **verifier** checks each element and signs a report. The report always states whether a limitations record is bound to the seal, whatever else it finds.

1.1 What this standard does not do

- **It does not make a record admissible.** Rules that let a qualified person certify a record or a copy (in the United States, Federal Rules of Evidence 902(13) and 902(14)) address authentication only. Hearsay, relevance, unfair prejudice and the right of confrontation are separate questions for the court. A package does not replace a witness where testimony is required.
- **It does not say that a record is true.** A package shows that a record is the one that was sealed, and what the machine itself reported about its reliability. A sensor can be wrong without reporting a fault.
- **It does not decide who is a qualified person.** A registry lists certifiers that the registry operator has found to meet stated criteria. Listing is not a finding of legal qualification. That is for the court.
- **It does not authorize release.** The package records the legal basis the custodian states. Whether release was lawful is for the court.

2. Conventions

- **Encoding.** All objects are JSON. A member name must not appear twice in one object, and a package with a repeated member name is rejected. Numbers are integers only; floating-point numbers are not allowed. Member names are ASCII. Integers must lie between -(2^53-1) and 2^53-1. Times are UTC in exactly the form `YYYY-MM-DDTHH:MM:SSZ` (zero-padded, capital Z); any other form is rejected. Every signed object has a `type` member that is a string without a line feed.
- **Canonical form.** Before hashing or signing, an object is serialized with members sorted by name, no insignificant whitespace, and UTF-8 text. This follows the approach of RFC 8785 for the restricted objects used here; where they differ, this standard governs.
- **Hash.** SHA-256. Hash values are written as 64 lowercase hexadecimal characters; uppercase or other forms are rejected. Firmware and model measurements are written `sha256:` followed by the hexadecimal hash. Salts and signature values are unpadded base64url in its canonical form (the text must equal the re-encoding of the bytes it decodes to); padded or non-canonical text is rejected.
- **Keys and signatures.** Ed25519 (RFC 8032). A key is written `ed25519:` followed by the 32-byte public key in unpadded base64url.
- **Signing.** To sign an object, remove its `sig` member, serialize it as above, prepend the bytes `machine-limits-standard/0.1/`, the object's `type` and a line feed, and sign the result. The signing prefix uses the raw `type` text. The `sig` member is `{"alg":"Ed25519","key":"…","value":"…"}`. A verifier accepts a signature only if it verifies under the key written in `sig.key`; whether that key is trusted is decided separately, as described in each section.
- **Object identity.** The hash of an object is the SHA-256 of its canonical form, signature included.

3. Records and salted field commitments

A **record** has an identifier and named fields. The sealing module commits to each field separately, so that a field can be withheld later while the record still proves into the seal.

For each field, the module chooses a secret random **salt** of at least 16 bytes (32 recommended) and computes:

```
field_commitment = SHA-256( 0x10 ‖ salt ‖ canonical([record_id, field_name, value]) )
```

The record's **leaf data** is the canonical form of `{"id": record_id, "fields": {name: field_commitment, …}}`, and its **leaf hash** is `SHA-256(0x00 ‖ leaf data)`.

A **released record** in a package has the form:

```
{"id": "ev-0002",
 "released": {"t": {"value": "…", "salt": "…"}, …},
 "withheld": {"operator_id": "<field commitment>"}}
```

A verifier recomputes the commitment of every released field from its value and salt, takes the commitments of withheld fields as given, rebuilds the leaf hash, and checks it against an inclusion proof (section 4). A withheld value cannot be recovered from its commitment without its salt. A reader can see that a field exists and its name; it cannot see its value.

4. The hash tree and the seal record

The sealing module places the leaf hash of every record it seals into an append-only binary hash tree built as in RFC 9162, section 2.1. The tree includes the module's limitations record (section 5) as one of its leaves.

The **seal record** is signed by the module and has these members:

Member	Meaning
`type`	`seal`
`version`	`0.1`
`module_id`, `module_key`	The module's identifier and public key.
`sealed_at`	Time of sealing.
`period`	`{start, end}`: the period the sealed records cover.
`tree`	`{alg: "rfc9162-sha256", size, root}`
`limitations`	`{record_id, index, leaf_hash}`: names the limitations record, its position in the tree, and its leaf hash.
`measurements`	`{firmware, model}`: `sha256:` measurements of the firmware and model running during the period.

A seal that does not name a limitations record is valid as a seal but fails the limitations check (V05).

An **inclusion proof** for a record is `{"index": n, "path": [hash, …]}`, checked as in RFC 9162, section 2.1.3.2, against `tree.root` and `tree.size`.

5. The limitations record

The limitations record is the module's signed list of the conditions it reports having detected during the period that makes the sealed records less reliable, with times, and a statement for each type of condition about whether the module monitored it. It is a record like any other (section 3), but it is always released in full: no field is withheld.

5.1 Condition types

Type	Meaning
`sensor_fault`	A sensor reported or showed a fault.
`sensor_stream_gap`	A sensor stream stopped producing data for a time.
`dropped_frames`	Frames or samples were dropped.
`clock_drift`	The module's clock differed from an outside time source by more than the module's stated tolerance.
`calibration_overdue`	A calibration was past its due date.
`firmware_change`	The firmware changed during the period.
`model_change`	The model changed during the period.

Type	Meaning
`records_deleted_or_not_made`	Records were deleted, or were not made when they should have been.

5.2 Fields

The record has exactly these four fields, and nothing else.

Field	Content
`module_id`	The module's identifier; must equal the seal's `module_id`.
`period`	Must equal the seal's `period`.
`coverage`	An object with exactly one entry for each of the eight types, each valued `detected`, `none_detected` or `not_monitored`.
`conditions`	A list of `{type, start, end, detail}`. Each condition must start no later than it ends, and lie within `period` (ends included).

`coverage` states each type outright. `detected` means at least one condition of that type is listed. `none_detected` means the module monitors that type and found none. `not_monitored` means the module does not monitor that type, so the absence of a listed condition says nothing about it. Every listed condition's type must be marked `detected`, and every type marked `detected` must have at least one listed condition.

5.3 Signature and binding

The limitations record is wrapped as `{"type": "limitations", "record": <released record>, "sig": …}` and signed by the module. The module signs the limitations record, so it is the module's own statement. A module that seals a second, cleaner record and names that one in its seal cannot be caught by this format. A limitations record is **bound** to the seal when all of the following hold: the seal names it; its record identifier and leaf hash match what the seal names; it proves into the sealed tree at the position the seal names; and its signature is by the module key.

6. Transparency registration

The seal record is registered with a transparency service, such as one built to RFC 9943, which returns a signed **receipt** proving that the seal existed in the service's log at a stated time. This version defines a simple receipt profile:

```
{"type": "receipt", "service_id": "…", "subject_type": "seal",
 "subject_hash": "<hash of the seal>", "logged_at": "…", "log_index": n, "sig": …}
```

A verifier accepts a receipt only if the service's key is in the verifier's trust inputs (section 13.1). Implementations should also support receipts in the form RFC 9943 defines; the profile for doing so is outside this draft.

7. Maker attestation chain

An **attestation** is signed by an issuer and names a subject key:

```
{"type": "attestation", "issuer_id": "…", "issuer_key": "…", "subject_key": "…",
 "subject_role": "sealing_module|intermediate", "issued_at": "…", "not_after": "…",
 "approved_measurements": {"firmware": ["sha256:…"], "model": ["sha256:…"]}, "sig": …}
```

The package carries a chain of one or more attestations. The first must be issued by a **maker key** in the verifier's trust inputs. Each later attestation must be issued by the previous attestation's subject. The last attestation's subject must be the module key. Every attestation except the last must have `subject_role` `intermediate`; the last must have `sealing_module`; a key whose role is `sealing_module` cannot issue attestations, and no key may appear twice in the chain. Measurements are `sha256:` followed by 64 lowercase hexadecimal characters. An inclusion proof may be given only for a released record or the limitations record. Each attestation's approved measurements must be among those approved by the one before it. Each attestation must be valid at `sealed_at`, and the last attestation's approved measurements must include the seal's firmware and model measurements. The measurements in the seal are the module's own statement; the chain shows only that a trusted maker key approved them for this module key.

8. Release record and legal-process record

The **release record**, signed by the custodian that released the records, states what was released and on what stated basis:

```
{"type": "release", "custodian_id": "…", "custodian_key": "…", "released_at": "…",
 "seal_hash": "…", "record_ids": ["…"],
 "basis": {"kind": "warrant|court_order|subpoena|party_consent",
           "reference": "…", "content_class": "machine_records|communications_content"},
"sig": …}
```

The **process-state record**, signed by the same custodian, states the state of that process when the records were released:

```
{"type": "process_state", "as_of": "…", "kind": "…", "reference": "…", "status": "…", "sig":
…}
```

Basis kind	Status that permits release in this format
`warrant`	`issued` or `served`
`court_order`	`issued` or `served`
`subpoena`	`objection_period_passed`
`party_consent`	`consent_given`

A subpoena is not treated as a warrant: release on a subpoena is shown only after the objection period has passed. `content_class` is required and must be `machine_records` or `communications_content`. Where it is `communications_content`, the basis must be a warrant or a party's consent. The `as_of` time must not be after `released_at`, and the process-state record's `kind` and `reference` must match the release basis. These are checks on what the custodian states; a verifier cannot establish from the package that a court order exists.

9. The evidence package

An evidence package is one JSON object:

Member	Content
`type`, `version`	`evidence_package`, `0.1`
`records`	The released records (section 3).
`inclusion_proofs`	A proof for each released record and for the limitations record, keyed by record identifier.
`limitations`	The limitations record (section 5).
`seal`, `receipt`	The seal record and its receipt.
`attestations`	The maker attestation chain.
`release`, `process_state`	The release and process-state records.
`certification`, `certification_receipt`	The certification (section 10) and the transparency receipt that registers it.
`registry_statement`	The certifier registry statement (section 11).
`corroboration`	Optional: `{record, receipt}` (section 12).

The **package core** is the object holding `records`, `inclusion_proofs`, `limitations`, `seal`, `receipt`, `attestations`, `release`, `process_state` and, when present, `corroboration`. A package with a member not listed in this section is rejected. A `corroboration` member that is present must hold exactly `record` and `receipt`. A package that has no limitations record is still a well-formed package; it fails V04 and V05.

10. Certification

A certifier signs a certification of the package core and states in it that the certifier is a qualified person:

```
{"type": "certification", "certifier_id": "…", "certifier_key": "…", "certified_at": "…",
 "package_digest": "<hash of the canonical package core>",
 "statements": {"process_produces_accurate_result": true,
                "copy_identified_by_digital_identification": true,
                "digital_identification_method": "…",
                "signer_attests_qualified_person": true}, "sig": …}
```

The statements follow the wording of the United States rules: that the records were generated by an electronic process or system that produces an accurate result, and that the copy is identified by a process of digital identification (here, by hash comparison against the sealed root). A hash comparison shows that the released fields match what was sealed; it does not show where the data was copied from. This JSON certification is not itself the written certification that Federal Rules of Evidence 902(13) and 902(14) require through Rule 902(11), which calls for a declaration under penalty of perjury and written notice to the other party; this format provides neither. Whether any certification satisfies a rule in a given proceeding is for the court. The `package_digest` must equal the hash of the canonical package core, so a certification cannot be reused for a package with different contents. The certification is registered with a transparency service as in section 6 (receipt `subject_type` `certification`), and the receipt's `logged_at` must

not be before `certified_at`. The time the certification was logged, not the time the certifier wrote in it, is the time used in section 11.

11. Certifier registry statement

A **certifier registry** lists certifiers that its operator has found to meet stated criteria, and records any suspension or revocation over time. A **registry statement**, signed by the operator, gives the listing history of one certifier key:

```
{"type": "registry_statement", "registry_id": "…", "as_of": "…", "certifier_key": "…",
 "events": [{"at": "…", "status": "listed|suspended|revoked"}], "criteria_ref": "…", "sig": …}
```

The status when the certification was logged is the status of the latest event at or before the receipt's `logged_at`; if several events share that instant, any status other than `listed` governs. It must be `listed`. The statement's `as_of` must not be before `logged_at`. Listing does not decide legal qualification.

12. Corroboration across machines

Where several machines sealed records of the same events, a compiler may compare them fact by fact and sign a **corroboration record**, which is registered with a transparency service (a receipt with `subject_type corroboration`):

```
{"type": "corroboration", "compiler_key": "…", "compiled_at": "…",
 "facts": [{"fact_id": "…", "description": "…", "reference_value": …,
            "expected_machines": ["…"],
            "observations": [{"machine": "…", "seal_hash": "…", "record_id": "…", "field":
"…", "value": …,
                              "observed_at": "…", "flagged_period": true|false}],
            "agree": ["…"], "disagree": [{"machine": "…", "delta": n|null}],
            "not_observed": ["…"], "in_flagged_period": ["…"]}], "sig": …}
```

The compiler signs with the key named in `compiler_key`. The standard does not establish who the compiler is; the agreement lists are the compiler's statement, checked only for consistency with its observations. Each observation names the `field` of the record that was observed. A machine may appear once per fact, and only if it is in `expected_machines`. For each fact: a machine **agrees** if its value equals the reference value, and **disagrees** otherwise (values are compared in canonical form, so `true` is not equal to `1`; with `delta`, the integer difference, where both values are integers; otherwise null). A machine in `expected_machines` that made no observation is in `not_observed`. A machine whose observation time falls within a listed condition's start and end (ends included, any condition type) is in `in_flagged_period`. A verifier recomputes all four lists from the observations and requires them to match. For an observation made by the package's own machine (the machine named in the seal's `module_id`), the verifier also requires that the observation name this package's seal, that the named record and field are released in the package with the same value, and that `flagged_period` matches the bound limitations record; if any of these cannot be checked, V12 fails. Corroboration is part of the package core when present, so removing or replacing it breaks the certification. A field that the package withholds cannot be corroborated, because the verifier cannot compare it. A corroboration record must contain at least one fact, the observation times must be valid times not after the verification, and `compiled_at` must not be after the corroboration receipt's `logged_at`, which must not be after `certified_at`.

13. The verifier

13.1 Trust inputs

A verifier is given, separately from the package, a trust file with three members: `maker_roots` (the maker keys it trusts), `transparency_services` (identifier to key), and `registry_operators` (identifier to key). Nothing in a package makes a key trusted. A verifier also needs a key of its own to sign its report; the report's signature identifies the verifier only if that key is known to the reader.

13.2 Checks

ID	Check
V01	The package is acceptable JSON (no repeated member names), has exactly the defined elements and version, and meets the encoding rules of section 2.
V02	The seal's type, version and tree algorithm are as defined, and its signature is valid and made by the module key it names.
V03	Each released record rebuilds to a leaf that proves into the sealed root, and the released records match the release record's list.
V04	A limitations record is present and signed by the module key.
V05	The limitations record is bound to the seal (section 5.3), is released in full, states every condition type, is internally consistent, and has the same period as the seal.
V06	The receipt is valid, from a trusted service, and covers this seal.
V07	The maker attestation chain is valid and approves the sealed firmware and model measurements.
V08	The release record is valid and tied to this seal, with a recognized basis.
V09	The process-state record permits release under the stated basis (section 8).
V10	The certification is valid, covers exactly the package core, makes the required statements, is dated no earlier than the release, and is registered by a trusted transparency service.
V11	The registry statement is valid, from a trusted operator, and shows the certifier as listed at certification time.
V12	If a corroboration record is present, it is valid, registered and internally consistent. Otherwise not applicable.
V13	The seal period starts no later than it ends and ends no later than sealing; then sealing, seal logging, release, certification

ID	Check
	and certification logging are in time order, and none is after the verification.

Malformed input in any element is a failure of the relevant check, not an error that stops the verifier. A verifier that cannot read a package at all reports `verification-incomplete`.

13.3 The report

The verifier signs a **verification report**:

```
{"type": "verification_report", "version": "0.1", "verifier_key": "…", "verified_at": "…",
 "package_digest": "<core>", "package_hash": "<whole package>",
 "checks": [{"id": "V05", "name": "…", "status": "pass|fail|not_applicable", "detail": "…"}],
 "limitations_record_bound": true|false,
 "result": "verification-complete|verification-incomplete", "notice": "…", "sig": …}
```

`limitations_record_bound` is always present and is true only if V04 and V05 both pass, whatever the other checks show. `result` is `verification-complete` only if every check passes or is not applicable. `verification-complete` describes the checks made. It is not a finding on admissibility, weight or truth, and the report's `notice` member says so.

14. Conformance

A producer conforms if the objects it produces satisfy sections 2 to 12. A verifier conforms if, given any package and trust file, it reports each of V01 to V13 as defined in section 13, states `limitations_record_bound` as defined there, and never reports `verification-complete` for a package that fails any check. A conformance suite, with sample packages that pass and sample packages that fail, accompanies this draft.

15. Security and privacy considerations

- **Salts.** A weak or reused salt lets a reader guess a withheld value. Salts must be random, secret and at least 16 bytes. The sample packages use fixed salts only so that they can be reproduced; a real module must not.
- **What withholding does not hide.** A reader learns that each withheld field exists, its name, and the record's position in the tree. A reader of the limitations record learns every condition the module reports.
- **The limitations record is the module's own statement.** A module can fail to detect a condition. The coverage field says which types it monitors; it cannot say that a monitored type was monitored well.
- **Clocks.** Times in a package are as stated by each signer. A report's `verified_at` is the verifier's own clock.
- **Keys.** The format identifies keys, not people or organizations. Binding a key to an identity is the job of the attestation chain, the registry and the verifier's trust inputs.
- **Custodian.** This version checks the custodian's signature but does not establish who the custodian is. A verifier user should decide separately whether to rely on the custodian.
- **Self-stated times.** The release time, process-state time and certification time are as the signer wrote them. The standard uses the logged time of the certification for the registry check, but not for the others.
- **Receipts.** The simple receipt profile in section 6 is for interoperability testing. Production use should rely on the RFC 9943 form.

16. What a passing result does and does not establish

A `verification-complete` report with `limitations_record_bound` true means: the released fields are the fields that were sealed; the module's own statement of its limitations is the one named in the seal and is in the same tree; the seal states firmware and model measurements that a trusted maker key approved for this module key; the seal was logged by a service the verifier trusts; the custodian states a basis for release that its stated process state supports; a certifier signed the statements in section 10 about the package core, and a trusted service logged that certification; and the registry statement shows the certifier as listed when it was logged. Fields that were withheld are not covered beyond their existence. A passing corroboration check means only that the compiler's statement is consistent with its own observations and, for this package's machine, with the released record; it does not establish who the compiler is.

It does not mean the records are accurate, that the module's list of conditions is complete or that the module was in fact running the stated software, that the custodian had authority, that the certifier is a qualified person, that the package is admissible, or that any witness is unnecessary.

17. References

- RFC 8032, Edwards-Curve Digital Signature Algorithm (EdDSA).
- RFC 8785, JSON Canonicalization Scheme (JCS).
- RFC 9162, Certificate Transparency Version 2.0 (hash tree and inclusion proofs, section 2.1).
- RFC 9943, An Architecture for Trustworthy and Transparent Digital Supply Chains.
- Federal Rules of Evidence 902(13) and 902(14).

Appendix A. Reference verifier and sample packages

The reference verifier runs offline: `python verify.py PACKAGE.json --trust TRUST.json`. It exits with status 0 only for `verification-complete`. The sample set contains two packages that pass and ten that fail. Five of the failing packages break the link between the limitations record and the seal: the record is missing; a different, validly signed record is substituted; the seal does not name a record; the record is not in the sealed tree; and the record leaves out a condition type. The others show a value changed after certification, a suspended certifier, a subpoena used before its objection period ended, an untrusted maker and an inconsistent corroboration record. A second set of fourteen failing cases and one passing control covers malformed encoding, repeated member names, extra members, a module key that attests other keys, a corroboration that contradicts the released record, an absent `content_class`, and ties and malformed times in the registry. The test runners run the failing cases first and confirm that each failing case fails before the passing cases are relied on; the one passing control in the second set is labeled as such. A third set of twenty-three cases, written by an independent reviewer who generated its own keys, covers further encoding, type, time, chain and corroboration cases. All names, times and records in the samples are fictional.