Machine Limits Standard

Draft 0.2 — a format for machine records and the limits the machine declares about them

Publisher: Machine Accountability Corporation
Published at: machineaccountability.org/standard
Status: Draft 0.2, November 10, 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 (licensing@machineaccountability.org). Free licenses are available for non-commercial, research, evaluation and conformance-testing use; the terms are at machineaccountability.org/standard/free-license.html.

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, how far 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. Member names are ASCII. Numbers are integers only, written as an optional minus sign followed by digits with no leading zero, no fraction and no exponent; `-0` is not allowed. A number written in any other form (for example `1.0`, `1e2` or `-0`) is rejected, even where its value is an integer. Integers must lie between -(2^53-1) and 2^53-1. Text that is not valid Unicode (for example a lone surrogate) is rejected.
- **Times.** Times are UTC in exactly the form `YYYY-MM-DDTHH:MM:SSZ` (zero-padded, capital Z) and must name a real moment of the Gregorian calendar: year 0001 to 9999, month 01 to 12, a day that exists in that month and year (29 February only in a leap year), hour 00 to 23, minute 00 to 59, second 00 to 59. `24:00:00` and leap seconds (`:60`) are not allowed. Any other form, or a time that does not exist, is rejected.
- **Types, versions and closed objects.** Every signed object has a `type` and a `version`. `type` is one of these ASCII values: `seal`, `receipt`, `attestation`, `release`, `process_state`, `certification`, `registry_statement`, `limitations`, `corroboration`, `verification_report`. (The package itself has type `evidence_package` and the hashed subject of a registry-event receipt has type `registry_event`; neither is signed as an object of its own, so neither is in this list.) `version` is the exact version of this standard (here `0.2`) and is covered by the signature. Every object defined in this standard has exactly the members listed for it: an object with a missing member or an unlisted member is rejected, so that no unsealed text can ride inside an object that is later certified. The one exception is `ext` (next item).
- **Extension member.** A signed object may carry one additional member, `ext`, an object whose content this standard does not define. It is covered by the signature and, if it is well-formed under the encoding rules, never changes the result of any check; an `ext` that breaks those rules (for example a fraction, or nesting deeper than 64 levels) fails V01 like any other part of the package. A verifier lists it in its report as “signer’s additional statements: not checked by this standard”. Objects that are not signed, and objects inside a signed object, do not carry `ext`.
- **Canonical form.** Before hashing or signing, an object is serialized with members sorted by name (by code point; names are ASCII), no insignificant whitespace, and UTF-8 text. A string is written between quotation marks. The quotation mark is written `\"` and the backslash `\\`. The characters U+0008, U+0009, U+000A, U+000C and U+000D are written `\b`, `\t`, `\n`, `\f` and `\r`. Any other character below U+0020 is written `\u` followed by four lowercase hexadecimal digits. Every other character, including `/`, U+007F and characters outside ASCII, is written as itself in UTF-8. For the restricted objects used here this is the same as RFC 8785; 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. The whole text of a hash, measurement or time must match its pattern: a value with a trailing line feed, a space or any other added character is rejected. 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/`, the object’s `type` and a line feed, and sign the result. The prefix names the major version only; during version 0 it is `/0/`, and it becomes `/1/` at version 1.0. The exact version is the signed `version` member. 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 exactly 32 bytes 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 released record has exactly the members `id`, `released` and, if any field is withheld, `withheld`; a released field has exactly `value` and `salt`; a salt of any length other than 32 bytes is rejected. A record may have a field named `t`, a time (section 2) of the event the record describes; the report uses it (section 13.3). 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`	`0.2`
`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.
`sig`	The module's signature.

Member	Meaning
`ext`	Optional (section 2).

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, …]}` with exactly those two members, where `n` is an integer (a boolean or any other value is rejected), 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 how far 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.
`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 of the form `{"monitoring": …, "unmonitored": […]}`.
`conditions`	A list of `{type, start, end, detail}`. Each condition must start no later than it ends, and lie within `period` (ends

Field	Content
	included).

`monitoring` is one of four values:

Value	Meaning
`full`	The module monitored this type throughout the period. `unmonitored` is an empty list.
`partial`	The module monitored this type except during the intervals listed in `unmonitored`.
`none`	The module did not monitor this type. `unmonitored` is an empty list.
`not_applicable`	This type cannot occur on this module, as the maker attests (below). `unmonitored` is an empty list.

`unmonitored` is a list of `{start, end}` intervals. It is not empty if and only if `monitoring` is `partial`. Each interval starts no later than it ends and lies within `period`; the intervals are in time order and do not overlap. Ends are included, so an interval that starts at the instant the previous one ends overlaps it and is rejected; write one interval instead. A condition may be listed only for a type whose `monitoring` is `full` or `partial`.

A reader must not take the absence of a listed condition as “checked and clean”. It says something only for a type that is `full`, and for a `partial` type only outside the unmonitored intervals; for `none` it says nothing, and for `not_applicable` there is nothing to say. The module cannot decide for itself that a type does not apply: the last attestation in the maker chain (section 7) carries `applicable_types`, the list of condition types that can occur on this module, and a type is `not_applicable` exactly when it is not in that list.

5.3 Signature and binding

The limitations record is wrapped as `{"type": "limitations", "version": "0.2", "record": <released record>, "sig": …}` (and optionally `ext`) 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", "version": "0.2", "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). `log_index` is an integer. The simple receipt is a **test profile**, for interoperability testing; a verifier labels it “simple receipt: test profile”

in the report's `receipt_profile` member and in the details of V06 and V10. From version 1.0, a receipt in the form defined by RFC 9943 is required, and a conforming verifier will not accept the simple form. Draft versions numbered 0.x accept it.

The receipt's `logged_at` is used in section 7 (the time at which attestations are tested) and in section 13.2 (V13).

7. Maker attestation chain

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

```
{"type": "attestation", "version": "0.2", "issuer_id": "…", "issuer_key": "…", "subject_key":
"…",
 "subject_role": "sealing_module|intermediate", "issued_at": "…", "not_after": "…",
 "approved_measurements": {"firmware": ["sha256:…"], "model": ["sha256:…"]},
 "applicable_types": ["…"], "sig": …}
```

`applicable_types` is a list, without repeats, of the condition types of section 5.1 that can occur on the module. It is present on the last attestation, the one whose role is `sealing_module`, and on no other.

The package carries a chain of one or more attestations, at most six. 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. 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.

Time. Each attestation must start no later than it ends. Each must be valid, ends included, at the seal's `sealed_at` and also at the `logged_at` of the seal's receipt (section 6); the module's own stated time alone is not enough. The period of each attestation after the first must lie within the period of the attestation that issued it.

Revocation. The verifier's trust inputs may list revoked keys, each with a revocation time (section 13.1). If any key in the chain, including the maker key, was revoked at or before the seal receipt's `logged_at`, the chain fails.

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", "version": "0.2", "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** states the state of that process when the records were released. It is signed either by the custodian or by a **process registry**, a body whose key is in the verifier's trust inputs (section 13.1) and that keeps the record of the process, such as a court:

```
{"type": "process_state", "version": "0.2", "signer_role": "custodian|process_registry",
 "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. With `signer_role` `custodian`, the signature must be the release record's custodian key, and the verifier reports the state as "custodian-stated". With `signer_role` `process_registry`, the signature must be a key in the trust inputs' `process_registries`, and the verifier reports the registry as the signer. The release record is signed by the custodian in either case. These are checks on what the custodian or registry 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`, `0.2`. The package itself is not signed.
`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", "version": "0.2", "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. The `statements` object has exactly the four members shown. 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. Each event in the history is itself registered with a transparency service, so that the operator cannot later change when an event happened:

```
{"type": "registry_statement", "version": "0.2", "registry_id": "…", "as_of": "…",
"certifier_key": "…",
 "events": [{"at": "…", "status": "listed|suspended|revoked", "receipt": <receipt>}],
 "criteria_ref": "…", "sig": …}
```

Each event's receipt has `subject_type` `registry_event` and a `subject_hash` equal to the hash of `{"type": "registry_event", "registry_id": …, "certifier_key": …, "at": …, "status": …}` (the values of this statement and event). It must come from a trusted transparency service, its `logged_at` must not be before the event's `at` and must not be after the statement's `as_of`. An event takes effect at its receipt's `logged_at`; the stated `at` does not decide when it took effect.

The status when the certification was logged is the status of the event with the latest effective time at or before the certification 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 the certification receipt's `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", "version": "0.2", "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, and its `seal_hash` must be a hash in the form of section 2. 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 and the difference lies within the integer range, and null otherwise). 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), or within an unmonitored interval of a type marked `partial` (section 5.2), 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 these members: `maker_roots` (the maker keys it trusts), `transparency_services` (identifier to key), `registry_operators` (identifier to key), `revoked_keys` (an object from key to the time it was revoked) and `process_registries` (a list of the keys of process registries it trusts). The last two may be empty or absent; if present they must have these shapes, and a verifier that cannot read them fails

V07 or V09 respectively rather than ignore them. 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, every object in it has exactly its defined members (and at most an `ext` where section 2 allows one), and it 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 the monitoring of every condition type as section 5.2 requires (including that `not_applicable` matches the maker's attestation), is internally consistent, and has the same period as the seal.
V06	The receipt is valid, from a trusted service, and covers this seal. The report labels a simple receipt as a test profile.
V07	The maker attestation chain is valid at the sealing time and at the logged time of the seal receipt, nests within the issuing attestations' periods, contains no key revoked by then, and approves the sealed firmware and model measurements. It cannot pass unless V06 has passed.
V08	The release record is valid and tied to this seal, with a recognized basis.
V09	The process-state record is signed by the custodian or a trusted process registry, as its `signer_role` states, and 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 (a simple receipt is labeled a test profile).
V11	The registry statement is valid, from a trusted operator, every event in it is registered by a trusted transparency service, and the events show the certifier as listed when the certification was logged.

ID	Check
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 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": "0.2", "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,
 "process_state_signer": "custodian|process_registry|null",
 "receipt_profile": "simple receipt: test profile",
 "limitations_summary": {<type>: {"monitoring": …, "unmonitored_intervals": n,
"conditions_listed": n, "reader_label": "…"}} | null,
 "conditions_against_records": {"conditions": [{"type": …, "start": …, "end": …, "detail": …,
                                                "released_record_ids_in_interval": ["…"]}],
                                "released_records_without_a_time": ["…"]} | null,
 "signers_additional_statements": {"note": "signer's additional statements: not checked by
this standard", "items": […]},
 "result": "verification-complete|verification-incomplete", "notice": "…", "sig": …}
```

`limitations_record_bound` is always present and is true only if V02, V04, V05, V06 and V07 all pass: the seal is the module's, the limitations record is the module's and is in the sealed tree, the seal was logged by a trusted service, and the maker chain approved the module. A limitations record cannot be reported as bound to a seal that failed any of these. `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.

`limitations_summary` gives, for each condition type, the monitoring value, the number of unmonitored intervals, the number of conditions listed, and a plain label for a reader ("none listed while monitored", "listed while monitored", "monitored only in part of the period", "not monitored", or "does not apply to this module"). It is null unless `limitations_record_bound` is true. `process_state_signer` names who signed the process-state record (section 8). `receipt_profile` states the receipt form the verifier accepted.

`conditions_against_records` lists every condition in the limitations record against the released records it covers. A released record is in a condition's interval, ends included, when the record has a `t` field (section 3) that is a time within it. `released_records_without_a_time` lists released records that have no readable `t`, which therefore cannot be placed against any condition. It is null unless `limitations_record_bound` is true. This listing places each declared condition against the times of the released records; it does not say that a record inside an interval is unreliable, or that

one outside it is reliable. `signers_additional_statements` lists any `ext` content found in the package, when V01 passes, and states that this standard does not check it.

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 exactly 32 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 a test profile, for interoperability testing. Production use should rely on the RFC 9943 form, which version 1.0 requires.
- **Revocation and time.** A package is tested at the logged time of its seal receipt, not only at the time the module wrote. Revocation is known to the verifier only through its trust inputs; a verifier that is not given a current revocation list cannot know of a later revocation.
- **Process-state signer.** A custodian-stated process state is the custodian's own statement about the process under which it released the records. A state signed by a process registry is only as reliable as that registry.
- **Additional statements.** Anything in an `ext` member is signed but is not checked by this standard, and a reader should not treat it as checked.

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, in objects that carry nothing besides what this standard defines (and any signed `ext` statements, which are not checked); 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 process state, signed as the report names, supports; a certifier signed the statements in section 10 about the package core, and a trusted service logged that certification; and the registry statement, each event of which was itself logged, shows the certifier as listed when the certification 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`. All names, times and records in the samples are fictional.

Sample packages. The sample set has thirteen packages: three that pass (P1, P2, P3) and ten that fail (F1 to F10). The download contains all thirteen and `trust.json`, and `build_samples.py` rebuilds them.

- Five failing packages break the link between the limitations record and the seal: the record is missing (F1); a different, validly signed record is substituted (F2); the seal does not name a record (F3); the record is not in the sealed tree (F4); the record leaves out a condition type (F5).
- The other five show a value changed after certification (F6), a suspended certifier (F7), a subpoena used before its objection period ended (F8), an untrusted maker (F9) and an inconsistent corroboration record (F10).
- `run_tests.py` runs the thirteen samples, the failing ones first, and confirms that each failing sample fails before the passing samples are relied on. The passing samples are labeled as controls.

Draft 0.2 cases. `run_draft02_cases.py` holds one hundred and one cases. Each builds a package, changes one thing, re-signs whatever depends on it, and states what a conforming verifier must do. Together they cover each change made in Draft 0.2: strict patterns, null delta, strict proof indexes, the bound flag, closed objects, attestation time and revocation, logged registry events, the signing prefix, the 32-byte salt, text forms, the coverage values, the receipt label, the listing of conditions against record times, the process-state signer, and the `ext` member. Further cases cover malformed encoding, repeated member names, 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.

Appendix B. Changes from Draft 0.1 (informative)

1. Patterns for hashes, measurements and times must match in full; a trailing line feed or space is rejected.
2. `delta` is null when the difference lies outside the integer range.
3. Inclusion-proof indexes must be integers.
4. `limitations_record_bound` is true only if V02, V04, V05, V06 and V07 pass.
5. Every defined object has exactly its listed members; signed objects may add an optional `ext`.

6. Attestations are tested at the logged time of the seal receipt as well as at sealing; periods nest; at most six attestations; trust inputs list revoked keys.
7. Each registry event carries a receipt; an event takes effect when it was logged.
8. The signing prefix is `machine-limits-standard/0/`; every signed object carries the exact `version`; `type` is one of a listed set.
9. The salt is exactly 32 bytes.
10. The integer text form, string escaping and the calendar validity of times are written out.
11. Coverage is stated as `full`, `partial` (with intervals), `none` or `not_applicable` (backed by the maker’s attestation); the words `detected`, `none_detected` and `not_monitored` are gone.
12. The simple receipt is labeled a test profile; the RFC 9943 form is required from 1.0.
13. The report lists each limitations condition against the times of the released records.
14. The process-state record is signed by the custodian or a trusted process registry, and the report names which.