KCS-ALP-L1: Agent Lockdown Profile

(ALP-L1)

Fail-closed interoperability profile for enterprise agent runtimes.

VERSION	1.0.0 (draft)
STATUS	Standards-track draft (normative interoperability language; not a certification)
RELEASE DATE	2026-02-18
CANONICAL URL	https://meridianverity.com/standards/
ZENODO DOI	https://doi.org/10.5281/zenodo.18682584

- Public-safe draft. Not a compliance certification.
- Non-binding unless incorporated by reference into a written agreement. Not legal advice.
- No patent license by publication.
- Release claims are evidence-based: procurement-grade status is defined by the signed integrity surface (see Section 10).
- Default safety posture is fail-closed: missing or unverifiable evidence HOLD/DENY.

Contents

Abstract

Tool-using agents are operators: they call APIs, move data, change permissions, and can execute destructive actions. In enterprise environments, “we have guardrails” is not a control unless you can prove enforcement. KCS-ALP-L1 (Agent Lockdown Profile — Level 1) is a *fail-closed interoperability profile* for agent runtimes. It standardizes the minimum gates and the minimum evidence surface so buyers and auditors can verify control offline (no uploads, no vendor-controlled verification). If prerequisites are missing or unverifiable, the correct state is HOLD, not a silent PASS. ALP-L1 standardizes:

- Fail-closed execution semantics (default HOLD/DENY on uncertainty)
- Signed allowlist-only tool/skill invocation (tamper-evident)
- Permit-before-send network egress (deny-by-default)
- Untrusted trusted boundary enforcement (no direct trigger of privileged actions)
- Secret isolation + scoped use (no plaintext secrets in prompts, tool output, or logs)
- High-risk action approval gates (HOLD until approval proof exists)
- Deterministic receipts + stable reason codes (a minimal registry snapshot)
- Version pinning + drift detection (baseline mismatch is evidence, not a surprise)
- Portable Evidence Packs with an integrity surface and offline verification + replay determinism Design intent: procurement-shaped. Everything required to validate a claim fits into a ticket: a pack ZIP, verifier output, and a 1–2 page audit summary. Receipts, not promises.

0. Status of this document

This is the public-safe standards-track draft of KCS-ALP-L1. Release gate (procurement-critical): This Zenodo record is a *procurement-grade signed release* if and only if the bundle-level integrity surface verifies offline:

1. sha256sum -c SHA256SUMS passes for every listed file, and
2. gpg --verify SHA256SUMS.asc SHA256SUMS yields a Good signature under the pinned artifact-signing fingerprint published on the canonical surface. If the required release-gate artifacts are missing, invalid, or unverifiable, consumers **MUST** treat this deposit as UNTRUSTED HOLD (fail-closed) for any conformance, safety, or procurement acceptance claim. See Section 10 and Appendix C.

- Canonical standards surface: https://meridianverity.com/standards/
- Immutable deposit: Zenodo record 10.5281/zenodo.18682584
- Scope: Agentic AI runtime / tool-using assistants / autonomous workflows in enterprise environments
- Safety posture: Fail-Closed (default HOLD/DENY unless verification passes)

- Profile boundary: This profile defines interoperability and evidence semantics for agent-runtime lockdown and offline verification. It does not certify compliance, safety, or fitness for purpose.
- Legal boundary: Non-binding unless incorporated by reference into a written agreement. Not legal advice. No patent license by publication.
- Integrity boundary: Artifacts are intended to be mirror-safe and verifiable offline using signed checksums and a pinned trust anchor fingerprint published on the canonical surface. Any authenticity failure, checksum failure, or fingerprint mismatch **MUST** be treated as UNTRUSTED HOLD (fail-closed).

0.1 Two-minute buyer-run demo (informative)

Procurement-grade claims fail when buyers cannot reproduce results. This deposit includes a practical offline verifier and portable fixture Evidence Packs so a buyer can confirm determinism in minutes. From bundle root:

```
# PASS baseline (valid pack; integrity surface present)
python3 verifier_contract/alp_l1_offline_verifier.py
  verifier_contract/fixtures/ALP_SAMPLE_PACK_TV-ALP-00
# HOLD (fail-closed): missing allowlist signature
python3 verifier_contract/alp_l1_offline_verifier.py
  verifier_contract/fixtures/ALP_SAMPLE_PACK_TV-ALP-00
# FAIL: digest mismatch (tamper / mismatch is evidence)
python3 verifier_contract/alp_l1_offline_verifier.py
  verifier_contract/fixtures/ALP_SAMPLE_PACK_TV-ALP-00
```

Expected overall_status values:

- TV-ALP-001 PASS
- TV-ALP-002 HOLD
- TV-ALP-005 FAIL Interpretation note (procurement-critical): Offline verifier PASS means “the pack conforms to the verifier contract and integrity/authenticity rules.” It does not mean the runtime executed a high-risk action. For example, a runtime that correctly returns HOLD for denied egress can still yield verifier PASS if the receipts and reasons match the expected contract. This deposit is designed to be attached to procurement and audit workflows, including:
- Normative profile specification (this document)
- A procurement/audit attachment template (docs/audit_summary_template.pdf)
- A verifier contract + portable fixture packs (verifier_contract/)
- A minimal reason-code registry snapshot (ALP_reason_code_registry_v1.0.0_PUBLIC_SAFE.json)
- Counsel-ready templates (templates/)

1. Purpose

1.1 Problem statement

Agent runtimes turn untrusted text into side effects. A prompt injection becomes a tool invocation. Tool sprawl becomes silent capability expansion. Network access becomes covert exfiltration. In most systems, the only “proof” of safety is a UI claim or a policy PDF. Procurement and security need something stronger:

- Deterministic receipts for every decision (ALLOW / DENY / HOLD), and
- portable evidence that can be verified offline under pinned baselines. If a buyer cannot reproduce the result offline, the buyer has a promise—not a control.

1.2 Goal

ALP-L1 defines the minimum runtime gates and evidence surface required to prove, offline, that:

1. Unsafe execution defaults to HOLD/DENY (fail-closed).
2. Tool use is constrained to a signed allowlist.
3. Network egress is deny-by-default and requires explicit permits.
4. Untrusted inputs cannot directly trigger trusted actions.
5. High-risk actions require explicit approvals.
6. Every decision produces receipts with stable reason codes.
7. Baselines are pinned; drift is detectable.
8. Evidence Packs are portable and offline-verifiable with replay inputs.

1.3 Relationship to KCS (informative)

ALP-L1 is designed to compose with the Kingdom Conformance Standard (KCS):

- KCS defines procurement-grade, offline-verifiable Evidence Packs, registry snapshots, and deterministic outcomes (PASS/FAIL/HOLD).
- ALP-L1 defines request-level verdicts (ALLOW/DENY/HOLD) and the minimum control points and artifacts required to prove enforcement. Recommended claim format:
- KCS L1 + KCS-ALP-L1 (offline verification + replayable vectors)
- KCS L2 + KCS-ALP-L1 (permit-before-commit at declared control points, if used)

2. Conformance language

The key words “**MUST**”, “**MUST NOT**”, “**SHOULD**”, “**SHOULD NOT**”, and “**MAY**” in this document are to be interpreted as normative requirements. A system conforms to ALP-L1 if and only if it satisfies all requirements labeled ALP-L1-REQ-*.

2.1 Outcome vs verdict (normative)

To avoid ambiguity, this specification uses two result layers:

- Verdict (runtime): ALLOW | DENY | HOLD
- Outcome (offline verification): PASS | FAIL | HOLD HOLD is fail-closed: it **MUST** be treated as "not verified" and **MUST NOT** be treated as PASS.

3. Terminology

- Agent runtime: The system executing an agent loop and invoking tools on behalf of an origin (human/service/agent).
- Tool/skill: Any callable capability that can cause side effects (I/O, network, file, external APIs, state changes).
- Allowlist: The current authorized tool/skill set and constraints, integrity-protected and verifiable.
- Permit: Offline-verifiable authorization for an otherwise-denied action class (egress destination/payload class).
- Approval gate: A control that requires explicit authorization for high-risk actions.
- Untrusted input: Content from external or user-controlled sources (web/email/chat/attachments) that may be adversarial.
- Receipt: A deterministic decision record emitted for every request and tool invocation.
- Evidence Pack: A portable directory (or ZIP) containing artifacts needed for offline verification and replay checks.
- Pin: A recorded digest/version/fingerprint that constrains what artifacts are accepted.
- Verifier: An offline tool that checks pack integrity/authenticity and reproduces expected outcomes under pinned baselines.

4. Profile overview (ALP-L1)

ALP-L1 assumes a modern agent runtime with:

- tool invocation
- network access
- potential access to secrets
- multi-origin requests (human/service/agent)
- enterprise audit requirements ALP-L1 constrains this runtime by enforcing fail-closed gates at critical control points:

1. Tool invocation (allowlist-only)
2. Network egress (permit-before-send; deny-by-default)
3. Trust boundary crossings (untrusted-to-trusted boundary)
4. Secret use (no plaintext exposure; scoped handles)
5. High-risk actions (approval required)
6. Evidence generation (receipts + packs + integrity)
7. Offline verification + replay determinism

4.1 Verdict semantics (normative)

- ALLOW: execution permitted
- DENY: execution blocked
- HOLD: execution not permitted until required evidence/approval/verification is provided (fail-closed pause)

5. Normative requirements (MUST)

Each requirement is specified as:

- Requirement ID
- **MUST** statement
- Minimum acceptance criteria

5.1 Fail-closed execution semantics

ALP-L1-REQ-001 (Fail-closed default) The implementation **MUST** default to HOLD (or DENY) when required verification inputs are missing, invalid, or unverifiable. Acceptance criteria:

- If any required artifact hash/signature check fails verdict **MUST** be HOLD/DENY and a reason code **MUST** be emitted.

5.2 Signed tool/skill allowlisting

ALP-L1-REQ-002 (Allowlist-only tools) The implementation **MUST** permit tool/skill invocation only when the tool/skill identifier is present in a current allowlist.
ALP-L1-REQ-003 (Signed allowlist) The allowlist **MUST** be integrity-protected via a verifier-recognized signature (eg, allowlist.sig) and the implementation **MUST** deny/hold execution if allowlist signature verification fails. Acceptance criteria:

- Unknown tool id DENY/HOLD with ALP1_TOOL_NOT_ALLOWLISTED
- Signature invalid/missing DENY/HOLD with ALP1_ALLOWLIST_SIG_INVALID

5.3 Network egress permit-before-send

ALP-L1-REQ-004 (Egress deny-by-default) The implementation **MUST** deny or hold outbound network transmission by default, unless an explicit permit exists for the destination and payload class. ALP-L1-REQ-005 (Egress evidence) For any permitted egress, the implementation **MUST** record destination, rationale, and policy reference in a receipt record. Acceptance criteria:

- Any outbound attempt without permit DENY/HOLD with ALP1_EGRESS_DENIED

5.4 Untrusted input boundary enforcement

ALP-L1-REQ-006 (Untrusted-to-trusted boundary) The implementation **MUST** enforce an explicit boundary between untrusted inputs (web/email/chat/attachments) and trusted actions (credential use, external sharing, destructive actions), such that untrusted inputs cannot directly trigger trusted actions without policy checks and, where required, approval gates. Acceptance criteria:

- Untrusted input triggering privileged action without gate DENY/HOLD with ALP1_UNTRUSTED_BOUNDARY_VIOLATION

5.5 Secret isolation and scoped use

ALP-L1-REQ-007 (Secret non-disclosure) The runtime **MUST NOT** expose plaintext secrets (API keys, tokens, passwords, signing keys) to agent prompts, tool outputs, or logs. ALP-L1-REQ-008 (Scoped secret use) When secrets are used, the implementation **MUST** enforce scope and expiry (least privilege) and record a non-sensitive reference (secret handle id) in the receipt. Acceptance criteria:

- Direct secret read/access outside scope DENY/HOLD with ALP1_SECRET_SCOPE_VIOLATION

5.6 High-risk action approval gate

ALP-L1-REQ-009 (Approval required for high-risk actions) The implementation **MUST** require an approval gate for high-risk actions, including at minimum: external sharing, permission changes, destructive actions, financial/commit operations, or policy changes. Acceptance criteria:

- High-risk action without valid approval token/receipt HOLD with ALP1_APPROVAL_REQUIRED

5.7 Deterministic verdict record + reason codes

ALP-L1-REQ-010 (Receipt for every decision) The implementation **MUST** emit a receipt record for every execution request, including ALLOW/DENY/HOLD decisions.
ALP-L1-REQ-011 (Reason codes) Each DENY/HOLD decision **MUST** include at least one

standardized reason code from the Reason Code Registry (Section 6). Acceptance criteria:

- Missing receipt non-conformant
- DENY/HOLD without reason code non-conformant

5.8 Drift / version pinning / change control

ALP-L1-REQ-012 (Pinned versions) The implementation **MUST** pin (record and enforce) versions/hashes for the policy bundle and allowlist used for decisioning. ALP-L1-REQ-013 (Drift detection) If a pinned artifact changes without an approved update process, the implementation **MUST** produce HOLD/DENY and record drift. Acceptance criteria:

- Policy hash mismatch HOLD/DENY with ALP1_POLICY_HASH_MISMATCH or ALP1_DRIFT_DETECTED

5.9 Evidence Pack generation + integrity

ALP-L1-REQ-014 (Evidence Pack required) The implementation **MUST** generate an Evidence Pack containing the minimum required artifacts defined in Section 7 for any evaluation session (or batch window). ALP-L1-REQ-015 (Integrity surface) The Evidence Pack **MUST** include a complete integrity surface (eg, SHA256SUMS and a detached signature SHA256SUMS.asc, or an equivalent verifier-supported mechanism). Acceptance criteria:

- Any missing required artifact verification **MUST NOT** yield PASS; runtime **MUST** treat the state as HOLD/DENY.

5.10 Offline verification + replay

ALP-L1-REQ-016 (Offline verification) A conforming implementation **MUST** support verification of an Evidence Pack in an offline (air-gapped) environment using only the pack contents plus pinned public keys. ALP-L1-REQ-017 (Replay determinism for verdict) The pack **MUST** include sufficient replay inputs to reproduce the verdict + reason codes for at least the included acceptance tests (where applicable), subject to redaction rules. Acceptance criteria:

- Offline verifier cannot validate signatures/hashes outcome **MUST** be HOLD with ALP1_ATTESTATION_INVALID or ALP1_INTEGRITY_CHECK_FAILED.
- Replay yields different verdict/reasons outcome **MUST** be FAIL with ALP1_REPLAY_MISMATCH.

6. Reason Code Registry (minimal)

6.1 Registry rules (normative)

- Format: ALP1_<TOKEN>
- Required fields in receipt: verdict, reasons[], policy_hash, allowlist_hash, timestamp
- Minimal registry rule: ALP-L1 implementations **MUST NOT** invent new ALP1_ codes without publishing an extension registry or using a vendor prefix (eg, VENDORX_...).

6.2 Codes (normative)

Legend: Typical verdict indicates expected runtime behavior when the reason applies.

Code	Typical verdict	Meaning (normative)
ALP1_PASS_CONFORMANT	**ALLOW**	Conformant; all required ALP-L1 checks passed under pinned baselines.
ALP1_ORIGIN_AUTH_FAILED	**DENY/HOLD**	Origin authentication/attestation failed or missing.
ALP1_POLICY_MISSING	**HOLD**	Required policy artifact not present.
ALP1_POLICY_HASH_MISMATCH	**HOLD/DENY**	Policy hash differs from pinned reference for this session.
ALP1_POLICY_SIG_INVALID	**HOLD/DENY**	Policy signature invalid/unverifiable.
ALP1_TOOL_NOT_ALLOWLISTED	**DENY/HOLD**	Tool/skill id not in allowlist.
ALP1_ALLOWLIST_SIG_INVALID	**HOLD/DENY**	Allowlist signature invalid/unverifiable.
ALP1_EGRESS_DENIED	**DENY/HOLD**	Outbound network transmission denied (no permit).
ALP1_UNTRUSTED_BOUNDARY_VIOLATION	**DENY/HOLD**	Untrusted input attempted to trigger trusted action without gate.
ALP1_SECRET_SCOPE_VIOLATION	**DENY/HOLD**	Secret access/usage exceeded scope/expiry or disclosure risk detected.
ALP1_APPROVAL_REQUIRED	**HOLD**	High-risk action requires approval token/receipt not provided.
ALP1_DRIFT_DETECTED	**HOLD/DENY**	Drift/change detected in pinned artifacts without approved update.

ALP1_INTEGRITY_CHECK_FAILED	**HOLD/DENY**	Evidence pack integrity surface failed (hash mismatch/missing file).
ALP1_ATTESTATION_INVALID	**HOLD/DENY**	Attestation envelope/signature invalid.
ALP1_REPLAY_MISMATCH	**HOLD/DENY**	Offline replay produced non-matching verdict/reason codes.
ALP1_QUORUM_NOT_MET	**HOLD/DENY**	Required authorization quorum not met (if used).
ALP1_RATE_LIMITED	**HOLD**	Safety throttling applied; execution deferred.

7. Evidence Pack format

7.1 File tree (canonical layout)

```
evidence-pack/
 manifest.json
 KEYS/                          # verifier-trusted public keys (optional but
   recommended)
 keys.pem
 keys.fingerprints.txt
 policy/
 policy.json
 policy.sha256
 policy.sig                   # optional if using separate signature; allowed by
   verifier
 allowlist/
 allowlist.json
 allowlist.sha256
 allowlist.sig
 receipts/
 receipts.jsonl               # newline-delimited receipt records
 attestation/
 attestation.dsse             # or verifier-supported envelope format
 integrity/
 SHA256SUMS
 SHA256SUMS.asc
 replay/
 README.md
 inputs/                      # minimal redacted inputs required for determinism
   checks
 expected/                    # expected verdict+reason snapshots (optional)
 verifier/
 verifier_output.json
 verifier.log                 # optional
 docs/
 audit_summary.pdf            # 1-2 pages, procurement/audit attachment
 README.md
```

7.2 Path and hash rules

- All paths in manifest.json **MUST** be relative to pack root.
- Hash algorithm **MUST** be SHA-256 unless verifier explicitly supports alternatives.
- SHA256SUMS **MUST** cover every file required by Section 7.1 (excluding directories).

7.3 Required artifacts (ALP-L1)

At minimum, an ALP-L1 Evidence Pack **MUST** include:

- manifest.json
- policy/policy.json + hash (and signature if used)
- allowlist/allowlist.json + signature
- receipts/receipts.jsonl
- integrity/SHA256SUMS + integrity/SHA256SUMS.asc (or equivalent)
- verifier/verifier_output.json
- docs/audit_summary.pdf (or an equivalent procurement-attachable summary)

8. Schemas

Schemas in this document are minimal required-field definitions. Extensions **MUST** be placed under an extensions object to stabilize the standard surface.

8.1 manifest.json schema (JSON Schema)

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "KCS Evidence Pack Manifest",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "pack_id": { "type": "string", "minLength": 8 },
    "profile_id": { "type": "string", "const": "KCS-ALP-L1" },
    "version": { "type": "string", "minLength": 3 },
    "created_at": { "type": "string", "format": "date-time" },
    "issuer": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "org": { "type": "string" },
        "name": { "type": "string" },
        "public_key_fingerprint": { "type": "string" }
      },
      "required": ["org", "public_key_fingerprint"]
    },
    "artifacts": {
      "type": "array",
      "minItems": 1,
      "items": {
```

```
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "path": { "type": "string", "minLength": 1 },
          "sha256": { "type": "string", "pattern": "^[a-f0-9]{64}$" }
        },
        "required": ["path", "sha256"]
      }
    },
    "policy_ref": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "path": { "type": "string" },
        "sha256": { "type": "string", "pattern": "^[a-f0-9]{64}$" }
      },
      "required": ["path", "sha256"]
    },
    "allowlist_ref": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "path": { "type": "string" },
        "sha256": { "type": "string", "pattern": "^[a-f0-9]{64}$" },

        "sig_path": { "type": "string" }
      },
      "required": ["path", "sha256", "sig_path"]
    },
    "integrity_ref": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "sha256sums_path": { "type": "string", "const": "integrity/SHA256SUMS" },
        "signature_path": { "type": "string", "const": "integrity/SHA256SUMS.asc" }
      },
      "required": ["sha256sums_path", "signature_path"]
    },
    "extensions": { "type": "object" }
  },
  "required": [
    "pack_id",
    "profile_id",
    "version",
    "created_at",
    "issuer",
    "artifacts",
    "policy_ref",
    "allowlist_ref",
    "integrity_ref"
  ]
}
```

8.2 allowlist.json (minimal schema)

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "ALP-L1 Tool Allowlist",
```

```
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "allowlist_version": { "type": "string" },
    "generated_at": { "type": "string", "format": "date-time" },
    "tools": {
      "type": "array",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "tool_id": { "type": "string" },
          "name": { "type": "string" },
          "risk_tier": { "type": "string", "enum": ["LOW", "MEDIUM", "HIGH"] },
          "egress_permits": {
            "type": "array",
            "items": { "type": "string" }
          }
        },
        "required": ["tool_id", "risk_tier"]
      }
    },
    "extensions": { "type": "object" }
  },
  "required": ["allowlist_version", "generated_at", "tools"]
}
```

8.3 Receipt record schema (receipts/receipts.jsonl)

Each line in receipts.jsonl is one JSON object conforming to:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "ALP-L1 Receipt Record",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "receipt_id": { "type": "string" },
    "request_id": { "type": "string" },
    "timestamp": { "type": "string", "format": "date-time" },
    "origin": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "origin_type": { "type": "string", "enum": ["HUMAN", "SERVICE", "AGENT"] },
        "origin_digest": { "type": "string" },
        "origin_attestation_ref": { "type": "string" }
      },
      "required": ["origin_type"]
    },
    "execution": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "tool_id": { "type": "string" },
        "action_type": { "type": "string" },
        "target": { "type": "string" }
      },
```

```
      "required": ["action_type"]
    },
    "policy_hash": { "type": "string", "pattern": "^[a-f0-9]{64}$" },
    "allowlist_hash": { "type": "string", "pattern": "^[a-f0-9]{64}$" },
    "verdict": { "type": "string", "enum": ["ALLOW", "DENY", "HOLD"] },
    "reasons": {
      "type": "array",
      "minItems": 1,
      "items": { "type": "string" }
    },
    "approval_ref": { "type": "string" },
    "egress": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "destination": { "type": "string" },
        "permit_id": { "type": "string" }
      }
    },
    "evidence_refs": {
      "type": "array",

      "items": { "type": "string" }
    },
    "extensions": { "type": "object" }
  },
  "required": [
    "receipt_id",
    "timestamp",
    "policy_hash",
    "allowlist_hash",
    "verdict",
    "reasons"
  ]
}
```

8.4 verifier_output.json (minimal schema)

Note (normative): overall_status **MUST** support HOLD to preserve fail-closed semantics.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Offline Verifier Output",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "pack_id": { "type": "string" },
    "verified_at": { "type": "string", "format": "date-time" },
    "overall_status": { "type": "string", "enum": ["PASS", "FAIL", "HOLD"] },
    "checks": {
      "type": "array",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "check_id": { "type": "string" },
          "status": { "type": "string", "enum": ["PASS", "FAIL", "HOLD"] },
          "detail": { "type": "string" },
```

```
          "reason": { "type": "string" }
        },
        "required": ["check_id", "status"]
      }
    },
    "vectors": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "passed": { "type": "integer", "minimum": 0 },
        "total": { "type": "integer", "minimum": 0 }
      }
    },
    "extensions": { "type": "object" }
  },
  "required": ["pack_id", "verified_at", "overall_status", "checks"]
}
```

9. Security considerations (non-normative)

ALP-L1 is designed for procurement and audit reality: controls must be tamper-evident, fail-closed, and independently verifiable offline.

9.1 Threats addressed (illustrative)

- Tool injection / capability sprawl: unvetted tools or skills invoked by an agent.
- Allowlist tampering: silent expansion by editing allowlists or downgrading signatures.
- Data exfiltration: outbound network transmission without an explicit permit.
- Prompt injection: untrusted content triggering privileged actions.
- Secret leakage: plaintext credentials appearing in prompts, tool outputs, or logs.
- Unauthorized high-risk actions: destructive or irreversible actions without explicit approval.
- Baseline drift / downgrade: policy/allowlist changes without approved change control.
- Evidence repudiation: inability to prove what was enforced at decision time.

9.2 Why fail-closed matters

False PASS is the catastrophic failure mode. ALP-L1 pushes uncertainty into HOLD:

- Missing or unverifiable signatures HOLD/DENY with stable reason codes.
- Missing evidence artifacts HOLD (not “best effort” ALLOW).
- Replay mismatch FAIL (determinism is part of the control contract).

9.3 Fast procurement red flags (informative)

- Verification requires uploading artifacts/logs to the supplier.
- Reason codes are free-form vendor strings (no pinned registry snapshot).
- “PASS” occurs when prerequisites are missing (not fail-closed).
- No test vectors / fixture packs (no buyer-run replay).
- Controls are described as policy statements without receipts proving enforcement.

9.4 What ALP-L1 does not do

ALP-L1 does not define model training, model architecture, or product UX. It does not replace a full security program. It defines the evidence and verification interface for runtime lockdown claims.

10. Registry & publication integrity (normative)

Receipts, not promises. A receipt is only as trustworthy as the publication rail that delivers it. This section defines the ALP-L1 release gate for mirror-safe artifacts, pinned trust anchors, and offline verification. It is written for standards-body and procurement use. Roles

- Registry Authority (RA): the publisher maintaining canonical ALP-L1 registries, schemas, and verifier contracts.
- Consumer: any buyer/auditor/verifier validating artifacts offline. Signed releases are governed by this section. If you intentionally publish an unsigned discussion preview, follow Appendix C. Consumers **MUST** treat unsigned previews as UNTRUSTED HOLD.

10.1 Canonical publication integrity (mandatory)

The RA **MUST** publish public ALP-L1 artifacts as mirror-safe bundles with: (a) SHA256SUMS (SHA-256 checksums), and (b) an authenticity mechanism for SHA256SUMS (eg, detached signature or DSSE envelope). The authenticity key fingerprint for public artifact signing **MUST** be pinned on the canonical standards surface (Canonical URL). Consumers **MUST** treat the pinned fingerprint as the trust anchor for artifact authenticity. Keyserver discovery or unauthenticated key distribution is insufficient. The RA **MUST NOT** publish artifacts that cannot be independently verified offline using the pinned fingerprint and the published authenticity mechanism.

10.2 Release signing gate (mandatory)

The RA **MUST** execute a release signing gate in a controlled RA environment that holds the artifact-signing private key. Private keys **MUST NOT** be distributed. A published

bundle **MUST** satisfy all of the following:

1. Content fixed before signing: the artifact set is finalized prior to checksum generation and signing.
2. Deterministic checksum inventory: SHA256SUMS is regenerated from the finalized artifact set (see 10.3).
3. Authenticity is verifiable: verify(SHA256SUMS.asc, SHA256SUMS) (or equivalent DSSE verification) succeeds as Good signature under the pinned fingerprint.
4. Integrity is verifiable: sha256sum -c SHA256SUMS (or equivalent) succeeds with no failures.
5. Consumer replay rehearsal: the RA **SHOULD** perform a clean-room verification rehearsal using only public materials (public key + bundle) and **MUST** treat any failure as UNTRUSTED HOLD. The RA **MUST NOT** publish a bundle when any step above fails.

10.3 Deterministic generation of SHA256SUMS (mandatory)

To prevent verifier disputes, the RA **MUST** generate SHA256SUMS deterministically.

- SHA256SUMS **MUST** include checksums for all files in the bundle except SHA256SUMS itself and its authenticity artifact(s) (eg, SHA256SUMS.asc).
- File paths **MUST** be normalized to a single canonical form (relative paths within the bundle).
- The inventory **MUST** be lexicographically ordered using a stable collation (eg, LC_ALL=C ordering).
- The RA **SHOULD** exclude common OS metadata files (eg, .DS_Store, Thumbs.db, __MACOSX/) from the bundle and from SHA256SUMS. (Informative example): set -euo pipefail export LC_ALL=C (find . -type f ! -name 'SHA256SUMS' ! -name 'SHA256SUMS.asc' -print0 | sort -z | xargs -0 sha256sum) > S sha256sum -c SHA256SUMS

10.4 Bundle-level authenticity artifact (mandatory)

If a detached signature is used, the authenticity artifact **MUST** be published as SHA256SUMS.asc and **MUST** verify against SHA256SUMS using the pinned fingerprint. The final release bundle **MUST** include:

- SHA256SUMS, and
- SHA256SUMS.asc (or DSSE envelope), and
- public key material (or pointer) necessary for offline verification (eg, an ASCII-armored public key file), provided that the pinned fingerprint remains the trust anchor. Critical invariants

- The signature **MUST** be generated over the exact SHA256SUMS file distributed in the final bundle.
- Any mismatch between a signed SHA256SUMS and the bundle contents **MUST** be treated as UNTRUSTED HOLD.

10.5 Record-level integrity (strongly recommended)

To prevent “ZIP extraction required before authenticity” disputes, the RA **SHOULD** publish a record-level checksum inventory and signature for the top-level deposited files (eg, the bundle ZIP and accompanying PDFs/MD). Recommended naming:

- RECORD_SHA256SUMS (checksums for Zenodo record files), and
- RECORD_SHA256SUMS.asc (detached signature over RECORD_SHA256SUMS). The record-level inventory **SHOULD** include at minimum:
- the bundle ZIP file checksum, and
- all normative/supporting PDFs, and
- this profile's publication integrity addendum, and
- (recommended) the pinned public key file for offline verification. This enables consumers to verify authenticity and integrity before extracting the bundle.

10.6 Consumer verification (offline; mandatory behavior)

Consumers **MUST** verify offline using the pinned fingerprint and the published authenticity mechanism. Minimum expected offline verification steps:

1. Obtain pinned fingerprint from the canonical standards surface.
2. Import the corresponding public key material (if provided).
3. Verify authenticity of SHA256SUMS (or DSSE envelope).
4. Verify integrity of all files with SHA256SUMS. Any authenticity failure, checksum failure, fingerprint mismatch, or unexpected file mutation **MUST** be treated as UNTRUSTED HOLD.

10.7 Key rotation and exceptions

Any change of artifact-signing keys **MUST** be explicit and **MUST** be announced as a new pinned fingerprint on the canonical standards surface. Silent key changes are prohibited.

10.8 Registry snapshots and change control (mandatory semantics)

Registries (reason codes, schemas, verifier contracts) are a stable interoperability API. Registry snapshots **SHOULD** be content-addressed and pinned by digest in receipts/packs, not by mutable URLs.

Meaning changes in-place are prohibited. Corrections **MUST** use explicit replacement + deprecation. Vendor-specific extensions **MUST NOT** replace standard ALP1_ *reason codes. If extensions are required, they **MUST** be published as an extension registry or use a collision-safe vendor prefix (eg, VENDOR_<ORG>_).*

10.9 Optional discovery rails (informative)

The RA **MAY** publish discovery surfaces (eg, REGISTRY_HEADS) to point to current registry snapshots. If published, they **MUST** be authenticated under the pinned artifact-signing key (or an explicitly pinned equivalent) and **MUST NOT** override pins carried inside receipts or packs. Heads are advisory pointers only. For monitoring integrity, heads **SHOULD** be append-only and hash-chained via prev_head_sha256 (see Appendix D). This deposit includes tools/registry_heads.py to generate and verify the chain deterministically.

Appendix A — Example receipt (informative)

```
{
  "receipt_id": "r_20260217_000041",
  "request_id": "req_9c2f",
  "timestamp": "2026-02-17T10:32:11Z",
  "origin": { "origin_type": "HUMAN", "origin_digest": "rohs:..." },
  "execution": { "tool_id": "tool.web.fetch", "action_type": "NETWORK_EGRESS",
    "target": "https://example.com" },
  "policy_hash":
    "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
  "allowlist_hash":
    "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb",
  "verdict": "HOLD",
  "reasons": ["ALP1_EGRESS_DENIED", "ALP1_APPROVAL_REQUIRED"],
  "evidence_refs": ["allowlist/allowlist.json", "policy/policy.json"]
}
```

Appendix B — Practical verifier acceptance vectors (informative)

This deposit includes portable fixture Evidence Packs and a reference offline verifier. The intent is simple: Buyers can run it offline. Determinism is part of the contract.

B.1 Fixture outcomes (PASS / HOLD / FAIL)

Fixture (pack ZIP)	Expected overall_status	What it proves
`TV-ALP-001 baseline`	**PASS**	Baseline pack validates; integrity surface + required artifacts present.

`TV-ALP-002 allowlist sigmissing`	**HOLD**	Fail-closed when allowlist authenticity is missing.
`TV-ALP-003 egressdenied hold`	**PASS**	Runtime HOLD for denied egress is valid when receipts/reasons match contract.
`TV-ALP-004 approvalrequired hold`	**PASS**	Runtime HOLD for missing approval is valid when receipts/reasons match contract.
`TV-ALP-005 digestmismatch`	**FAIL**	Tamper / mismatch is detected and cannot be waved through.
`TV-ALP-006 replaymismatch`	**FAIL**	Replay determinism is enforced; divergence is evidence.

B.2 Where the contract lives

- vectors_KCS-ALP-L1_v1.0.0.json — contract-level vectors (PASS/FAIL/HOLD rules)
- verifier_contract/test_vectors.json — which fixture packs to run
- verifier_contract/fixtures/ — the portable packs (ZIPs)
- verifier_contract/alp_l1_offline_verifier.py — reference verifier (informative)
  Procurement note: This is not “trust our UI.” This is a buyer-run replay kit.

Appendix C — UNSIGNED PREVIEW publication labeling rules (normative)

Receipts, not promises. Drafts are allowed. Confusion is not. An UNSIGNED PREVIEW is a discussion-only publication where one or more required authenticity steps of the ALP-L1 release gate are not satisfied (eg, detached signatures are not present or not verifiable under the pinned fingerprint). An unsigned preview can be useful for review, but it **MUST** be impossible to mistake for a procurement-grade release. Unless otherwise stated, the requirements in this appendix apply to the publisher / Registry Authority (RA).

C.1 Normative requirements

ALP-L1-PUB-UNS-001 (Machine-detectable label) If a publication is an unsigned preview, the RA **MUST** include the exact token UNSIGNED_PREVIEW (uppercase, underscore) in all of the following entry points:

- Zenodo record title (prefix is **RECOMMENDED**)
- Zenodo record description (first 10 lines)

- Bundle root README.md (first 20 lines)
- Whitepaper front matter (cover or first page) Acceptance: a consumer can determine preview status by reading only the title/description/README without opening any bundle contents. ALP-L1-PUB-UNS-002 (Fail-closed consumer expectation) The RA **MUST** state that consumers **MUST** treat an unsigned preview as UNTRUSTED HOLD (fail-closed) for any conformance, safety, or procurement acceptance claim. Acceptance: the description banner contains both UNSIGNED_PREVIEW and the phrase EXPECT HOLD (case-insensitive). ALP-L1-PUB-UNS-003 (No conformance claims) For an unsigned preview, the RA **MUST NOT** claim “ALP-L1 conformant”, **MUST NOT** claim procurement-grade verification, and **MUST NOT** imply a PASS outcome for release-gate integrity/authenticity. Acceptance: no conformance claim appears in the title/description/README; any verification language is explicitly scoped to “discussion-only preview”. ALP-L1-PUB-UNS-004 (No signature-shaped placeholders) For an unsigned preview, the RA **MUST NOT** publish files that *appear* to be release-gate authenticity artifacts unless they are valid and verifiable. In particular, the RA **MUST NOT** include:
- SHA256SUMS.asc
- RECORD_SHA256SUMS.asc
- DSSE envelopes or witness receipts that cannot be verified offline under the pinned fingerprint If the RA wishes to ship “what will exist at release time”, it **MUST** do so as templates (eg, *.TEMPLATE,* .EXAMPLE) that cannot be mistaken for real signatures. Acceptance: a consumer running gpg --verify SHA256SUMS.asc SHA256SUMS does not encounter a misleading “signature file present but invalid” situation, because the file is absent (or explicitly a template, not named as a real signature). ALP-L1-PUB-UNS-005 (No record-level checksum inventories under production names) If the RA publishes record-level integrity materials for illustration in an unsigned preview, they **MUST** be published as templates that cannot be mistaken for real inventories produced over Zenodo record bytes. In particular, the RA **MUST NOT** publish RECORD_SHA256SUMS in an unsigned preview unless it is a real inventory generated from the published record files. Templates **MUST** be named with a non-ambiguous suffix (eg, RECORD_SHA256SUMS.TEMPLATE). Acceptance: a consumer **MUST NOT** be able to mistakenly run sha256sum -c RECORD_SHA256SUMS against a placeholder file and believe record-level integrity was performed. ALP-L1-PUB-UNS-006 (Non-incorporation boundary) An unsigned preview **MUST NOT** be incorporated by reference into an agreement and **MUST NOT** be used as the referenced conformance contract for procurement acceptance. Acceptance: the description banner includes “NOT FOR INCORPORATION BY REFERENCE” (or equivalent) and the bundle README includes a “Non-binding preview” boundary section. ALP-L1-PUB-UNS-007 (Upgrade path to signed release) When the RA publishes a signed release that satisfies Section 10, the RA **MUST**:
- publish it as a new immutable deposit version (new Zenodo version)

- include a change log entry describing the transition from preview to signed release
- update the canonical standards surface with the pinned fingerprint and pointers
- mark the unsigned preview record as “superseded” in its description (link to the signed release) Acceptance: the preview record contains an explicit pointer to the signed release record.

C.2 Required banner text (copy/paste)

The following banner text is **RECOMMENDED** and is sufficient to satisfy the requirements above when placed at the top of the Zenodo description and in README.md: UNSIGNED_PREVIEW — NOT A RELEASE — NOT PROCUREMENT-GRADE. EXPECT HOLD. This deposit is provided for discussion/review only. Release-gate authenticity artifacts are not present/verifiable under the pinned fingerprint. Do not incorporate by reference. Do not claim ALP-L1 conformance based on this preview.

Appendix D — REGISTRY_HEADS discovery rail template (informative)

Some consumers want a single “what is current?” pointer for monitoring. ALP-L1 supports an optional discovery rail named REGISTRY_HEADS. Receipts, not promises. Heads are advisory pointers only. They **MUST NOT** override pins carried inside receipts/packs.

D.1 Format (recommended)

- File name: REGISTRY_HEADS (bundle root)
- Encoding: UTF-8
- Wire format: JSON Lines (one JSON object per line)
- Each line is a head record conforming to schemas/registry_heads.schema.json

D.2 Append-only + prev_head_sha256 chain

If published, the RA **SHOULD** treat REGISTRY_HEADS as append-only: never rewrite earlier records; only append a new record. Chain rule (deterministic):

- For head record N > 1: prev_head_sha256 **MUST** equal sha256(canonical_json(head_{N-1})).
- canonical_json is JSON with keys sorted, separators (',', ':'), encoded as UTF-8 bytes. This provides cheap, offline tamper evidence and reduces equivocation risk. It is a monitoring rail, not a verification rail.

D.3 RA rail (tooling)

This deposit includes a reference RA tool:

- tools/registry_heads.py
- config: tools/registry_heads_config_ALP-L1.json From bundle root: python3 tools/registry_heads.py append --verify-after python3 tools/registry_heads.py verify --check-files RA best practice: run this step before generating SHA256SUMS, so REGISTRY_HEADS is covered by the release signing gate.

D.4 Included materials

- Schema: schemas/registry_heads.schema.json
- Single-record template: templates/REGISTRY_HEADS_ALP-L1.template.json
- Single-record example: templates/REGISTRY_HEADS_ALP-L1.example.json
- JSONL example file: templates/REGISTRY_HEADS_ALP-L1.example.jsonl

Appendix E — Operational extension rails (informative)

ALP-L1 is a runtime lockdown profile. It deliberately stays minimal and interoperable. However, procurement reality rewards rails that turn “we will do X” into “here is the evidence that we did X.” This appendix defines optional operational rails that can be shipped as artifacts and verified offline. These rails are not required for ALP-L1 conformance unless a party explicitly incorporates them by reference. They are provided to reduce three common procurement gaps:

1. incident reporting and response,
2. evals that actually gate releases, and
3. ecosystem/variant tracking (especially for open-weight distribution).

E.1 Incident reporting rail (optional)

Recommended artifacts (templates included in templates/ops/):

- incident_report_ALP-L1.json (or .jsonl for multiple incidents)
- incident_taxonomy_ALP-L1.md (classification + mapping guidance)
- response_playbook_ALP-L1.md (HOLD / revoke / replace / deprecate rail) If an incident report is published as part of a record, the publisher **SHOULD**:
- make it append-only (do not rewrite incident history), and
- authenticate it under the same pinned artifact-signing trust anchor used for release artifacts, and
- reference affected artifacts by digest (hashes), not mutable URLs.

E.2 Eval gate coupling rail (optional)

Recommended artifacts (templates included):

- eval_plan_ALP-L1.json — which vectors/evals gate which releases (pinned versions)
- eval_results_ALP-L1.json — signed results summary + receipts/pack pointers
- release gate rule: “missing/failed required evals HOLD” (fail-closed) Intent: a benchmark is not a control unless it gates shipping.

E.3 Ecosystem forensics & takedown rail (optional)

Recommended artifacts (templates included):

- heritage_fingerprint_ALP-L1.json — artifact lineage/derivation hints (content-addressed)
- ecosystem_watchlist_ALP-L1.json — known variants and their status (allowed/blocked/deprecated)
- takedown_notice_template_ALP-L1.md — platform-ready notice template Intent: make post-release monitoring and response procedural and evidence-shaped, not ad-hoc.

References (informative)

- Meridian Verity Group, *Kingdom Conformance Standard (KCS) v1.0.0* (public-safe standards-track draft), canonical surface: https://meridianverity.com/standards/
- Meridian Verity Group, *KCS Procurement Quick Reference v1.0.0* (procurement-grade offline-verifiable checklist)
- Meridian Verity Group, *KCS Publication integrity addendum* (release signing gate; deterministic checksums; mirror-safe publication)