LARRI
VERIFICATION PROTOCOL

Volume V of V

LARRI Implementation and Operations Guide

Version 0.1.0

Purpose. Informative operational guidance for establishing verification context, issuing receipts, constructing prepared-directory bundles, controlling external trust, running deterministic verification and conformance, interpreting bounded results, assembling release evidence, and correcting or reissuing immutable artifacts. The guide describes public protocol operation without prescribing private decision methods or software architecture.

Property	Value
Document type	Informative implementation and operations guide
Publication set	LARRI 0.1.0, Volume V of V
Protocol	LARRI - Ledger of Attested Records and Receipts for Integrity
Version	0.1.0
Status	Experimental
Use boundary	Synthetic laboratory and supervised evaluation only
Publication date	27 July 2026
Official source	https://larriverification.org

Assurance boundary. LARRI explains and verifies defined post-attestation integrity and attribution properties under explicit public contracts and independently supplied trust. It does not establish application truth, completeness, human or organizational identity, authority, independently trusted time, legal sufficiency, regulatory compliance, production readiness, or permission to act.

Informative guide

Contents

Document relationship. Volumes I through IV are normative. This guide is informative: it cannot create, amend, waive, or resolve a protocol requirement. Exact structures and identifiers are governed by the approved schemas and registries; frozen vectors and expected bytes govern concrete conformance comparison. Reference code, examples, deployment preferences, and this guide are not normative authority.

1. Purpose, authority, and operating principles

LARRI - Ledger of Attested Records and Receipts for Integrity - separates application-defined result production from public verification. An Issuer/Core implementation may use private evidence, policies, rules, models, thresholds, prompts, or other application-specific methods. LARRI constrains the public artifacts and verification behavior, not the private method that produced the application assertion.

The guide is intended for implementers, issuers, attestors, verification operators, conformance-harness operators, publishers, reviewers, and relying processes working within the Experimental synthetic-laboratory and supervised-evaluation boundary.

1.1 Informative authority boundary

- Volume I controls assurance meaning, roles, trust boundaries, verification targets, dependency order, result meaning, and non-goals.
- Volume II controls exact fields, lexical forms, canonical bytes, signature messages, external-trust objects, resource limits, prepared-directory bundle rules, refusal representation, and normalized-result bytes.
- Volume III controls optional profile and extension semantics.
- Volume IV controls conformance vectors, frozen evidence, claims, reports, publication gates, and release-envelope behavior.
- When wording in this guide appears inconsistent with one of those sources, the normative source controls and this guide should be corrected.

1.2 Public and private boundaries

Publicly fixed by LARRI	Outside the base verification boundary
Artifact structures; canonicalization; identity calculations; digest and signature messages; trust treatment; target and profile selection; checks; refusal precedence; status meanings; normalized result bytes; conformance evidence and report rules.	Decision mathematics; policy logic; private evidence; models or weights; prompts; thresholds; internal data structures; orchestration; source layout; programming language; storage design; deployment architecture; product workflow.

Operating principle. Treat every received artifact as untrusted. Establish semantic contracts, target, profile context, external trust, configuration, and public limits explicitly. Preserve signed and committed bytes exactly. Interpret PASS and REFUSE only within the complete recorded context.

1.3 Assurance limits

- A valid signature attributes bytes to the supplied public key; it does not identify or authorize a person or organization.
- A commitment establishes correspondence to identified bytes; it does not establish correctness, completeness, legality, or availability.
- A PASS establishes only the selected LARRI integrity assurance. It does not reproduce or approve the private process that produced the assertion.
- A REFUSE means the selected assurance was not established. It is not a finding of fraud, falsity, negligence, or substantive invalidity.
- Neither PASS nor REFUSE directly authorizes a relying-party action.

2. Roles and operation context

2.1 Logical roles

Role	Operational responsibility	Boundary
Core Protocol	The public normative protocol, principally Volume I.	It is not the product-side decision engine.
Issuer/Core implementation	Completes an application-specific process, forms an assertion, and coordinates artifact generation.	Its private decision method need not be disclosed or reproduced.
Issuer	Forms the public assertion wrapper, commitments, limitations, signer declaration, receipt, and bundle controls.	The issuer does not become trusted merely by creating the artifact.
Attestor	Controls or invokes a private signing key and produces a registered attestation.	The private decision engine need not possess the signing key.
Operator	Selects semantic contracts, target, profiles, artifact or subject, external trust, configuration, and limits.	Those selections are protocol-visible inputs.
Verifier	Performs only public deterministic verification behavior.	It does not require private Core logic, private evidence, generator state, or network services.
Conformance harness	Supplies controlled public inputs and compares exact frozen expected bytes and exit classes.	Reference implementation output is not an oracle.
Publisher or claim owner	Assembles reports and release evidence and controls correction, withdrawal, and reissue.	Publication authentication does not enlarge conformance scope.
Relying process	Interprets a bounded result in a separately authorized process.	LARRI does not make the substantive decision.

Roles may be combined operationally. Combining them does not collapse the protocol boundaries: trust remains independent of the artifact, verification remains reproducible from public rules, and relying-party authorization remains separate.

2.2 Explicit operation context

Input	Operational record
Semantic contracts	The exact closed semantic-contract package and independently selected contracts_sha256.
Verification target	Exactly receipt, bundle, or profiled-bundle after registry resolution.
Profiles and extensions	Explicit selected profile identifiers, exact combination where applicable, required extensions, and result contract.
Artifact or subject	Complete receipt bytes or a prepared-directory logical bundle subject.
External trust	The complete independently supplied external-trust object.
Configuration	The exact closed verification-configuration object.
Resource limits	The published base record and any valid stricter selected values.

2.3 Invocation establishment

Target-scoped verification exists only after the semantic contracts and an enabled target have been established. Two pre-target cases use the closed invocation-refusal object:

Invocation code	Operational meaning
SEMANTIC_CONTRACTS_NOT_ESTABLISHED	The semantic-contract root, its closed manifest, listed files, or selected contracts digest could not be established. contracts_sha256 is null.
UNSUPPORTED_VERIFICATION_TARGET	Semantic contracts were established, but the selected target token did not resolve to one enabled target. contracts_sha256 is present.

Target-scoped configuration defects. After an enabled target exists, a missing, malformed, schema-invalid, non-stricter, or otherwise invalid configuration is represented as a target-scoped verification REFUSE. It is not converted into an invocation refusal and no fabricated verifier result is emitted.

3. Configuration, limits, and identity availability

3.1 Configuration identity

Condition	configuration_sha256
Exact baseline configuration	null
Target-scoped refusal before any valid configuration is selected	null
Selected valid nonbaseline stricter configuration	SHA-256 of the exact canonical configuration bytes; retained on every later PASS or REFUSE, even when no stricter threshold is reached.

The configuration input is evaluated under the published bootstrap safety boundary. A selected override does not govern parsing of the configuration object that selects it.

3.2 Six base limits

Limit	Published inclusive maximum	Operational meaning
protocol_json_max_bytes	5,242,880	Maximum bytes in one protocol-owned JSON input.
json_max_depth	64	Maximum container nesting depth; the top-level value has depth 1.
json_max_object_members	200,000	Maximum total object-member count in one JSON input.
json_max_array_length	200,000	Maximum number of elements in any one array.
bundle_max_entries	10,000	Maximum discovered regular-file and fixed-control entries in a prepared-directory subject.
bundle_max_total_bytes	524,288,000	Maximum total materialized bytes under the prepared-directory subject.

Each published maximum is inclusive. The immediately greater value follows the exact refusal or execution classification defined by the normative contracts and frozen evidence.

3.3 Digest availability

Field	Becomes available when	Null or empty condition	Retention
contracts_sha256	Semantic contracts are established.	No target-scoped result exists before this point.	Present in every target-scoped result.
artifact_sha256 - receipt	The complete finite receipt byte stream was obtained.	Receipt bytes were incomplete, or safe execution ended before a truthful full-stream digest could be computed.	Retained through later decoding, schema, trust, identity, signature, commitment, or profile refusal.
artifact_sha256 - prepared directory	The required preceding stages through external trust pass and the manifest establishes a complete canonical logical subject with matching manifest_id.	The manifest is missing, incomplete, over limit, malformed, noncanonical, unresolved, or identity-mismatched.	Retained through later path, file, receipt, attestation, commitment, or profile refusal.
trust_input_sha256	The complete trust JSON strictly decodes and canonicalizes.	Trust is missing, incomplete, over limit, or cannot be canonicalized.	Retained through later trust-schema, duplicate-label, key, fingerprint, permission, signature-use, or downstream refusal.
configuration_sha256	A valid nonbaseline configuration is selected.	Exact baseline, or refusal before any valid configuration is selected.	Retained through every later result branch.

No fabricated identity. Zero digests, empty-input digests, fixed placeholders, inferred directory hashes, and other synthetic substitutes are not used. A nullable field is null only on the exact branch where the identity cannot be truthfully computed.

4. Receipt intake and issuance

4.1 Immutable receipt intake

- Receive the finite receipt byte stream and preserve the exact bytes before decoding or repair.

- Record the received-byte digest when the complete stream is available, even when strict decoding or a later check refuses.
- Use read-only source bytes and a separate controlled working area for parsing and verification where practical.
- Do not normalize line endings, repair JSON, change Unicode, edit signatures, or transform the receipt to make it pass.
- Treat every transformed or corrected object as a different artifact with a different byte identity.

4.2 Public issuance sequence

1. The application completes its private or application-specific process.
2. The issuer forms the closed public assertion wrapper, identifying its application namespace and type.
3. The issuer creates typed commitments over exact bytes or bytes produced by a registered canonicalization.
4. The issuer records every applicable registered limitation.
5. The issuer creates the signed signer declaration and selects the registered signature policy.
6. The issuer computes the nonrecursive receipt identity over the canonical signed content with receipt_id omitted.
7. The issuer inserts receipt_id, canonicalizes the final signed content, and computes its SHA-256 digest.
8. The attestor applies the exact registered signature suite and returns the closed attestation object.
9. The issuer preserves the final immutable receipt bytes and records their received-byte identity before exchange.

4.3 Baseline signature construction

Exact suite meaning. larri/ed25519-sha256-v1 applies Ed25519 to the raw 32-byte SHA-256 digest produced from the final canonical signed content. It is neither direct-message Ed25519 nor RFC 8032 Ed25519ph. A different message construction, post-quantum scheme, or hybrid construction uses a different registered suite and policy identifier.

4.4 Time and private material

- created_at is a signed issuer assertion. Integrity does not make it independently trusted time.
- asserted_at is optional unauthenticated envelope metadata under the baseline attestation and has no trusted-time effect.
- A digest may bind private policies, models, thresholds, reports, or evidence without publishing them. A digest is not encryption and low-entropy values may be guessable.
- Confidentiality, access control, and encryption are applied outside the base protocol.

5. Prepared-directory bundle construction

Base subject. The base bundle subject is a prepared directory governed by the fixed controls manifest.larri.json and manifest.larri.sig.json. Base LARRI does not define another carrier form or a canonical directory serialization. Any other carrier form exists only through a separately registered Volume III carrier extension.

5.1 Construction sequence

1. Finalize every in-scope payload file in its immutable form.
2. Validate safe normalized relative paths and reject collisions, links, non-regular entries, and unsafe names.
3. Assign content_role and artifact_type explicitly for each manifest payload entry.
4. Compute the exact byte length and SHA-256 digest of each final payload file.
5. Create manifest.larri.json with closed scope and one entry for every in-scope non-control file.
6. Exclude manifest.larri.json and manifest.larri.sig.json from the manifest payload listing.
7. Compute the nonrecursive manifest identity with manifest_id omitted, insert the value, and compute the final canonical manifest digest.
8. Create manifest.larri.sig.json under the registered manifest-attestation contract.
9. Preserve the final controls and payload files without in-place modification.

5.2 Manifest entry responsibilities

Member	Operational meaning
path	Location only: a safe normalized relative path. It does not select receipt or extension semantics.
content_role	The registered byte-preparation and verification treatment.
artifact_type	The declared protocol-owned type, permitted application-namespaced type, or null opaque classification.
byte_length	Exact count of the file bytes listed by the manifest.
sha256	SHA-256 over those exact file bytes.

5.3 Artifact classification

- A contained receipt is discovered only when artifact_type == "larri/decision-receipt".
- Path, filename, suffix, prefix, directory placement, media type, and content sniffing do not select receipt or profile semantics.
- A registered artifact type is checked for compatibility with its content role.
- Unknown or unselected application-namespaced extension files may receive generic manifest path, length, and digest integrity treatment. They do not execute extension checks, satisfy a profile, authorize a claim, or count as extension conformance.
- Unknown protocol-owned artifact types follow the registered refusal behavior rather than being treated as application content.

5.4 Logical subject identity

For a prepared directory, artifact_sha256 identifies the established logical manifest subject. File checks determine whether the observed directory realizes the declared subject. The result does not invent a snapshot hash for the directory and does not imply that another carrier has the same bytes.

6. External trust and key operations

External trust is external to the artifact, not necessarily external to the organization. A single organization may prepare and control the trust input offline, provided the artifact cannot establish or replace the trust used to verify itself.

6.1 Base trust entry

Field	Operational purpose
key_id	Unique opaque key label within one external-trust object.
suite_id	Enabled signature suite for the supplied key bytes.
public_key_encoding	The registered encoding; base Ed25519 uses base64-raw.
public_key	Canonical encoding of the exact key bytes.
fingerprint_algorithm	Base value sha-256.
fingerprint	SHA-256 over the exact decoded public-key bytes.
permitted_targets	Duplicate-free target identifiers in verification-target registry order.

6.2 Offline trust preparation

1. Obtain the exact public-key bytes and expected fingerprint through an operator-controlled channel independent of the artifact exchange.
2. Encode the key exactly as required by the selected registered suite.
3. Recompute or independently confirm the fingerprint over the exact raw key bytes.
4. Assign a unique key_id and the exact permitted target set.
5. Construct the complete closed trust object, canonicalize it, and retain its identity when canonicalization succeeds.
6. Protect the trust object and target permissions against unauthorized substitution.

6.3 Trust-use phase boundary

Phase	Trust handling
Stage 4	Validate the complete external-trust object and the key needed for a direct receipt or the manifest control attestation. Unused entries are not decoded as keys merely because they are supplied.
Stage 9	After manifest attestation and file integrity have passed, resolve and use the trust entry needed for each contained receipt attestation. No unverified bundle file is read early to establish Stage 4 trust.

trust_fingerprints contains only keys that actually satisfied required signatures. Merely supplied, rejected, or unused entries remain absent from that array.

6.4 Rotation and status limits

- Rotation selects a new explicit trust context and key identifier; it does not silently reinterpret an existing artifact.
- The base protocol does not define a live certificate authority, revocation service, network lookup, regulator, or external institution.
- Identity governance, authority, revocation, compromise-effective time, transparency, and long-term validation require separately specified evidence or profiles.
- Private keys, seeds, recovery material, and live credentials stay outside artifacts, contracts packages, vectors, logs, and support bundles.

7. Verification operations

Verification is a deterministic evaluation of explicit public inputs. Internal architecture, programming language, concurrency model, storage strategy, and deployment form may differ, provided observable behavior matches the normative contracts and frozen evidence.

7.1 Eleven-stage order

Stage	Name	Operational objective
1	Input and subject safety	Establish the enabled target context and applicable limits; prevent unsafe processing.
2	Strict decoding and schema recognition	Apply strict decoding and identify the intended registered top-level schema.
3	Contract and feature resolution	Resolve profiles, exact combinations, required extensions, artifact types, checks, additional trust contracts, result contracts, and refusal codes.
4	External trust	Validate the complete trust object and direct receipt or manifest-control key eligibility.
5	Manifest canonical form and identity	For bundle targets, validate canonical form, reject declared duplicate paths, and recompute manifest identity and digest.
6	Detached manifest attestation	Verify the manifest attestation only after Stage 5 passes.
7	Closed bundle scope and file integrity	Validate observed paths, closed scope, artifact_type, content role, exact length, and digest.
8	Receipt identities and signed bindings	Strictly process required receipt files and recompute identities and signed duplicate relationships.
9	Contained-receipt attestations	Resolve contained-receipt key use and apply the exact suite and policy.
10	Commitments and selected profiles	Recompute required commitments and evaluate conditions belonging to records already resolved at Stage 3.
11	Normalized result	Select the deterministic refusal state where applicable and emit the exact closed result.

7.2 Dependency behavior

- Profile selection and required-extension resolution occur at Stage 3. Stage 10 does not rediscover profiles, merge records, or select alternatives.
- Manifest identity precedes manifest signature; manifest signature precedes file integrity.
- Bundle file integrity precedes contained-receipt trust use.
- A failed, missing, or unsafe prerequisite blocks dependent work. The corresponding downstream check is SKIP rather than being executed merely to collect another refusal.
- Safely completed sibling refusals follow the public refusal-registry order.

7.3 Controlled execution

- Supply artifacts, semantic contracts, trust, configuration, and profiles through distinct controlled channels.
- Use read-only inputs and a fresh controlled working scope for each operation.
- Keep core verification offline and provide every required input explicitly.
- Separate human diagnostics from canonical protocol output.

- Retain the exact artifact or logical subject identity, contracts digest, trust-input identity where available, actual used fingerprints, configuration identity where applicable, and canonical normalized result.
- Vary non-protocol environment details during testing so paths, locale, time zone, process identity, or scheduling cannot affect canonical bytes.

8. Profiles and result interpretation

8.1 Current base profile state

No optional profile is enabled. profiled-bundle remains a defined enabled target, but the base optional-profile, profile-combination, and optional-extension registries contain no enabled optional assurance. Without an enabled selected profile, Stage 3 refuses with PROFILE_SELECTION_REQUIRED. No positive profiled-bundle PASS path is manufactured. Positive profile guidance becomes applicable only through a real companion profile with complete semantic contracts, trust treatment, result traceability, and conformance evidence.

Future lineage, time, witness, transparency, privacy-screening, carrier, multisignature, post-quantum, hybrid, and domain assurances are extension capacities rather than active base assurances.

8.2 Check statuses

Status	Interpretation
PASS	The check executed and its requirement was satisfied.
REFUSE	The check executed and produced a blocking protocol refusal.
SKIP	A prerequisite failed or safe evaluation was blocked.
NOT_PRESENT	An optional subject was absent and the selected context did not require it.
UNVERIFIED	A signed assertion, commitment, or optional attestation was present, but the supplied material did not allow recomputation or evaluation.

8.3 Result context

Component	Interpretive use
Target and profiles	The exact selected assurance operation. Receipt and bundle use an empty profiles array.
Artifact or subject identity	The received receipt bytes or established prepared-directory logical manifest subject, when computable.
Semantic contracts	The exact contracts_sha256 governing the operation.
Trust identity and actual use	trust_input_sha256 identifies the supplied canonical trust value when available; trust_fingerprints lists only successfully used required keys.
Limits and configuration	The base limit record and a non-null digest for every selected valid nonbaseline configuration.
Outcome and refusals	PASS or REFUSE, deterministic primary refusal, and safely completed sibling refusals.
Checks and limitations	Fixed-order check statuses and every registered qualification that remains applicable.

8.4 Distinct claims

Property	What it establishes
Artifact verification PASS or REFUSE	Evidence about one selected receipt, bundle, or profiled-bundle operation.
Protocol Conformance PASS or FAIL	Evidence that one implementation reproduced public LARRI behavior for one exact reported scope.
Clean-Room Reproduction	An optional claim that separately engineered implementation behavior reproduced the reported scope.
Independent Interoperability	An optional claim about identified independent implementations and exchanged inputs.
Reproducible Release	An optional claim about rebuilding an implementation artifact; it is not bundle-carrier reproducibility.
Release-envelope authentication	Authentication of exact canonical release-manifest bytes under the supplied key.

Bounded language. None of these properties establishes artifact truth, application correctness, completeness, signer identity or authority, independently trusted time, legal sufficiency, regulatory compliance, production readiness, or permission to act.

8.5 Suggested result language

PASS. The selected LARRI checks completed for the identified subject, target, contracts, profile context, limits, configuration, and external trust. The statement retains every applicable limitation.

REFUSE. The selected LARRI assurance was not established. Review the primary refusal and public check statuses. The result is not a substantive finding about the application assertion.

9. Privacy, logging, and retention

9.1 Canonical output and diagnostics

- Normalized results exclude raw private evidence, sensitive matched values, private keys, seeds, credentials, absolute host paths, hostnames, process identifiers, local timestamps, execution duration, stack traces, stdout, stderr, temporary paths, and free-form exception text.
- Operational logs remain separate from canonical protocol output and do not alter its bytes.
- Public detail codes are closed and registered; internal diagnostics may be richer but remain outside normalized comparison.
- Log identifiers, public counts, ruleset names, and report commitments only when their disclosure is appropriate for the operating context.

9.2 Privacy-screening boundary

Privacy-screening evidence may establish only that identified checks were completed over an identified byte or artifact scope and produced an identified bounded result. It never proves universal absence of sensitive information. Inaccessible, encrypted, unsupported, and unparsed content remains explicitly qualified by the selected profile.

9.3 Retention

Retention periods are organizational choices outside base LARRI. Materials that may support reproducibility and investigation include exact artifacts, prepared-directory controls and payloads, semantic contracts, trust input, configuration, canonical normalized results, frozen conformance evidence, detailed executions, gate reports, conformance reports, release controls, and non-sensitive operational records.

Confidentiality. Digest commitments do not provide confidentiality. Protect private source material and retained evidence with application-appropriate encryption, access controls, logging controls, and disposal procedures outside the base protocol.

10. Conformance execution

10.1 One-way identity layers

Layer	Identity	Dependency
Semantic contracts	contracts_sha256	Canonical semantic-contract manifest and its closed semantic files only.
Conformance evidence	vector_set_sha256	May depend on contracts_sha256, semantic-record digests, vectors, fixtures, frozen expected bytes, result contracts, exit classes, and applicability.
Conformance report	Digest of canonical report bytes	May depend on the exact scope, both earlier identities, detailed executions, and the G01-G13 gate report.
Release envelope	Digest of final canonical release manifest	May inventory earlier identities, reports, normative PDFs, publication metadata, and release payload files.

Dependencies flow only in that order. Semantic records do not contain evidence or release identities, and conformance evidence does not depend on the release envelope.

10.2 Evidence generation and execution

1. Establish the approved semantic-contract sources and exact closed semantic manifest.
2. Compute contracts_sha256.
3. Generate immutable vectors, fixtures, and frozen expected outputs from the public rules.
4. Build the deterministic aggregate evidence index and compute vector_set_sha256.
5. Execute every applicable vector at least twice and execute the complete suite in a shuffled order.
6. Emit detailed executions in aggregate evidence-index vector order.
7. Execute G01 through G13 and emit one canonical gate report ordered G01 through G13.
8. Produce the conformance report from the exact scope, execution counts, detailed-report digest, and G01-G13 gate-report digest.

Expected outputs are frozen artifacts. Ordinary conformance execution compares against them and does not regenerate its own oracle.

10.3 Deterministic arrays

Array	Duplicate rejection and order
requirement_ids	Exact ASCII duplicate comparison; exact ASCII order.
profiles and extensions	Applicable registry rank, then exact ASCII identifier.
target_contexts	Verification-target registry rank, then canonical context bytes.
platform_scope	Exact ASCII descriptor.
vectors	Exact ASCII vector_id.
semantic_record_digests	Semantic class, applicable registry rank, then identifier.
artifact inputs, fixtures, expected outputs, and release payload files	normalized_path.
detailed executions	Aggregate evidence-index vector order.
gate results	Exactly G01 through G13.

Canonical JSON preserves array order; it does not sort arrays. Generators therefore reject duplicates and emit the public order before canonicalization.

10.4 Vector-count equation

Count	Treatment
applicable_vector_count	Total number of applicable evidence-index vector positions.
Successful expected classes	verification_pass_count + verification_refuse_count + invocation_refuse_count + execution_limit_count
Vector-execution failures	comparison_failure_count + harness_error_count
Required equation	applicable_vector_count equals the sum of the six counts above.
publication_gate_failure_count	Separate count of failed G01-G13 gates. Gate failures are not vector executions and do not enter applicable_vector_count.

10.5 Harness controls

- Supply artifacts, semantic contracts, trust fixtures, and configuration through separate harness-controlled channels.
- Treat vectors, fixtures, expected outputs, and contracts as read-only.
- Run each vector in an isolated mutable scope and reset state between vectors.
- Disable undeclared network access.
- Record implementation subject, platform descriptor, contracts digest, vector-set digest, trust-fixture digest, and configuration digest for each execution.
- Classify artifact-caused refusals as protocol data, not as harness crashes or successful early termination.

11. Release assembly and final audit

11.1 Release controls

The fixed release control files are release.larri.json and, when supplied, release.larri.sig.json. Neither control appears in the release-manifest files array. files inventories every other release payload exactly once in normalized_path order, with its registered role, byte length, and SHA-256 digest.

release_id is nonrecursive: canonicalize the closed release manifest with release_id omitted, compute SHA-256, encode lowercase hexadecimal, insert the value, and canonicalize the final manifest.

An optional detached release signature authenticates the exact final canonical release.larri.json bytes under one registered signature contract. It does not authenticate an inferred directory serialization or another carrier representation.

11.2 Final sequence

1. Complete the detailed execution report.
2. Complete the canonical G01-G13 gate report.
3. Complete the conformance report that binds that G01-G13 report.
4. Assemble release.larri.json with every non-control release payload.

5. Optionally assemble release.larri.sig.json over the exact final canonical release manifest.
6. Execute G14 last over the exact final package.
7. Permit publication only after G14 passes.

No cycle. G14 is not embedded in the conformance report, gate report, release manifest, detached release signature, or another artifact that G14 audits. Its outcome remains outside the audited package. A G14 failure blocks publication.

11.3 Authentication limit

Release authentication proves only that the verified key authenticated the exact canonical release-manifest bytes. It does not repair failed evidence, turn FAIL into PASS, enlarge conformance scope, establish organizational authority, or authorize a relying-party action. No outside reviewer, regulator, or live service is required for the base publication process.

12. Correction, withdrawal, and reissue

A claim owner withdraws a Protocol Conformance PASS when a discovered vector, comparison, harness, traceability, security, privacy, identity, G01-G13, or G14 defect would invalidate the bound claim or block publication.

12.1 Correction sequence

1. Preserve the existing authenticated report and package bytes.
2. Identify and correct the authoritative public source material.
3. Regenerate affected semantic-contract and conformance-evidence identities.
4. Rerun the complete vector suite, including repeated and shuffled execution.
5. Rerun G01 through G13.
6. Generate new detailed, gate, and conformance reports.
7. Assemble a new release manifest and optional detached release signature.
8. Execute G14 last over the new exact package.
9. Issue new immutable bytes and digests only after G14 passes.

Immutable evidence. Authenticated artifacts, reports, manifests, and signatures are not edited in place. A targeted partial rerun does not restore PASS. Correction creates new bytes, new identities where affected, and a complete new evidence trail.

13. Unsupported inputs and safe handling

This guide describes the current 0.1.0 contract only. Unsupported protocol, schema, canonicalization, suite, policy, target, profile, extension, carrier, or result identifiers follow the registered refusal behavior. They are not interpreted through aliases, punctuation substitution, case folding, heuristic negotiation, content sniffing, or implementation-private fallback.

Opaque application content remains data. It is parsed or executed only when a selected registered contract requires that behavior and all prerequisites have passed.

An implementation may expose local diagnostics, operator controls, APIs, or deployment integrations, but those features remain outside canonical protocol output and do not change the public LARRI behavior.

Stage boundary. The guide does not generate or modify schemas, registries, requirement inventories, vectors, fixtures, frozen expected results, reports, gates, semantic-contract identities, evidence identities, release manifests, or signatures.

A. Illustrative domain-neutral workflows

These workflows are informative sequences. They do not add fields, identifiers, claims, checks, or implementation modules.

A.1 Receipt issuance

Step	Action
1	Complete the private application process and select the public assertion.
2	Form the closed assertion wrapper and typed commitments.
3	Record limitations and the signer policy.
4	Compute receipt_id nonrecursively, then canonicalize and hash the final signed content.
5	Obtain the registered attestation from the attestor.
6	Preserve the exact final receipt bytes and record their byte identity.

A.2 Receipt verification

Step	Action
1	Establish semantic contracts, enabled receipt target, configuration, and external trust.
2	Preserve the complete receipt bytes and record artifact_sha256 when available.
3	Run the applicable stages: safety, strict decoding, contract resolution, trust, receipt identity and binding, receipt attestation, commitments, and normalized result.
4	Interpret PASS or REFUSE with the target, identities, checks, and limitations.

A.3 Prepared-directory bundle construction

Step	Action
1	Finalize payload bytes and safe normalized paths.
2	Assign content_role and artifact_type and compute exact length and SHA-256.
3	Create the closed manifest, excluding the two fixed controls from files.
4	Compute manifest_id and the final canonical manifest digest.
5	Create the detached manifest attestation and preserve the complete prepared directory.

A.4 Prepared-directory bundle verification

Step	Action
1	Establish bundle target, contracts, configuration, and Stage 4 trust.
2	Validate manifest canonical form and identity, then the detached manifest attestation.
3	Validate closed scope, paths, artifact types, content roles, lengths, and digests.
4	Process required contained receipts and use contained-receipt trust only at Stage 9.
5	Evaluate commitments and resolved profiles, then emit the normalized result.

A.5 No-profile profiled-bundle refusal

Step	Action
1	Establish semantic contracts and the profiled-bundle target.
2	Resolve the selected profile context at Stage 3.
3	Because no optional profile is enabled, emit the contract-resolution refusal with PROFILE_SELECTION_REQUIRED.
4	Do not invent a base profile, fall back to bundle, or report a positive profiled-bundle PASS.

A.6 Conformance execution

Step	Action
1	Establish contracts_sha256 and vector_set_sha256.
2	Run every applicable vector at least twice and in a shuffled complete-suite order.
3	Compare exact exit classes and canonical bytes against frozen expected evidence.
4	Emit detailed executions in evidence-index order and execute G01-G13.
5	Generate the gate report and conformance report.

A.7 Release assembly and G14

Step	Action
1	Complete detailed, G01-G13 gate, and conformance reports.
2	Assemble release.larri.json and optional release.larri.sig.json.
3	Execute G14 last over the exact final package.
4	Keep G14 output outside the package it audits and publish only after G14 passes.

A.8 Correction and reissue

Step	Action
1	Preserve the existing authenticated bytes and withdraw an invalidated claim.
2	Correct the public source and regenerate affected identities and frozen evidence.
3	Rerun the complete suite and G01-G13; generate new reports.
4	Assemble a new release envelope, execute G14 last, and issue new immutable bytes.

B. Operational checklists

The checklists are informative reminders. Their controlling requirements remain in Volumes I through IV.

B.1 Operation context

[ ] Semantic contracts and contracts_sha256 are established.

[ ] The selected target is enabled and exact.

[ ] Profile, combination, extension, and result-contract selections are explicit.

[ ] The artifact or prepared-directory subject is fixed before processing.

[ ] External trust and the exact configuration are supplied independently.

B.2 External trust

[ ] Public-key bytes, suite, fingerprint, key_id, and permitted targets are exact.

[ ] The trust object arrived through an operator-controlled channel independent of the artifact.

[ ] Duplicate key labels are absent.

[ ] Only actually used required keys enter trust_fingerprints.

[ ] Private keys, seeds, and credentials are absent from artifacts and logs.

B.3 Issuance

[ ] The assertion wrapper is closed and application-scoped.

[ ] Commitments bind exact or registered canonical bytes.

[ ] Limitations remain explicit.

[ ] receipt_id is computed with its member omitted.

[ ] The attestor applies the exact registered suite and policy.

[ ] Final receipt bytes are immutable and retained.

B.4 Bundle preparation

[ ] Every payload is final before manifesting.

[ ] Paths are safe normalized relative paths.

[ ] content_role and artifact_type are explicit and compatible.

[ ] Every payload appears exactly once with exact length and digest.

[ ] The two fixed controls are excluded from files.

[ ] The manifest identity and detached attestation are complete.

B.5 Verification

[ ] The eleven-stage order is preserved.

[ ] Stage 3 resolves profiles, combinations, extensions, and result contract.

[ ] Manifest identity precedes manifest signature.

[ ] File integrity precedes contained-receipt trust use.

[ ] Blocked dependent checks are SKIP.

[ ] Human diagnostics remain outside canonical output.

B.6 Result interpretation

[ ] Target, profiles, contracts, trust, limits, and configuration context are visible.

[ ] Nullable identities follow the exact Volume II branches.

[ ] Limitations remain in user-facing summaries.

[ ] PASS is not described as truth or authority.

[ ] REFUSE is not described as fraud or substantive invalidity.

B.7 Conformance execution

[ ] Expected outputs are frozen and read-only.

[ ] Every applicable vector is executed at least twice and in a shuffled suite.

[ ] Detailed executions follow evidence-index order.

[ ] The vector-count equation balances.

[ ] G01-G13 are complete before the conformance report.

B.8 Release assembly

[ ] Detailed, G01-G13 gate, and conformance reports are final.

[ ] release.larri.json and optional release.larri.sig.json are fixed controls.

[ ] The controls are excluded from files and every other payload is listed once.

[ ] release_id is computed nonrecursively.

[ ] G14 executes last and remains outside the audited package.

B.9 Correction and reissue

[ ] Existing authenticated bytes are preserved.

[ ] The authoritative public source is corrected.

[ ] Affected identities and evidence are regenerated.

[ ] The complete suite and G01-G13 are rerun.

[ ] New reports and a new release envelope are produced.

[ ] G14 passes before new publication.

B.10 Privacy and secrets

[ ] Canonical output contains no private evidence or host diagnostics.

[ ] Operational logs are separate from protocol output.

[ ] Privacy-screening claims remain bounded to identified checks and scope.

[ ] Digest commitments are not represented as encryption.

[ ] No private keys, seeds, credentials, or raw sensitive matches are published.

End of guide.