

SOLA Standard for Classified Data Code and Brick Compositions

Draft 0.3 for technical review

SOLA working draft

2 October 2026

Contents

Scope and normative language	2
Implementation baseline	2
Classification of every artifact	2
Flow algebra	3
LLM deployment and security profiles	4
Security levels within each dimension	5
Managed API and on premises comparison	5
Inference admission and composed proofs	6
Brick contracts	6
Recursive compositions	6
Enforcement and policy governance	7
Controlled exceptions	7
Assurance levels	8
Proof obligations for combinations	8
Worked combination	9
Automatic documentation from Java contracts	9
Reference generation pipeline	10
Security metadata and proof evidence	10
Conformance and adoption	11
Sources and evidence boundaries	11

This standard proposes one security model for SOLA data, executable code, reusable bricks and recursively composed systems. Every artifact has a classification. Every executable capability has a contract. Every composition must demonstrate that its connections preserve the policy. The highest assurance level requires a machine checked proof for the exact composition and an explicit argument connecting that model to its implementation.

This is a new design draft, informed by the maintained SOLA v5 library, a locally available AWS Micro v4 implementation, the commons custody standard and OpenAPPA. It is not an adopted standard, an OpenAPPA implementation or a certification of the existing fleet. The formal rules below are proposed SOLA extensions. No composition proof has yet been produced.

Scope and normative language

MUST and MUST NOT identify conformance requirements. SHOULD identifies a recommendation whose exceptions must be documented. MAY identifies an optional feature. Conformance always names the standard version, deployment profile, artifact revisions, policy and assurance level.

The system covers records, schemas, prompts, source, dependencies, binaries, configurations, models, generated patches, proofs, logs, storage, tool calls, interprocess messages and external destinations. A deployment profile MUST name which channels it mediates. Unmediated channels prevent claims that depend on their mediation.

The model separates three questions: who may receive an artifact, what authority its contents may exercise, and what the implementation has actually demonstrated. A confidential artifact is not automatically trustworthy; public code can be trustworthy without being entitled to private records.

Implementation baseline

The maintained entry point is `sola-master/library/cells/sola-zero-v5`. Its optional bricks and named profiles establish a useful separation between a small cell and selected capabilities. The v5 backend uses key plus external-factor authentication and retains verified change candidates separately from deployment. Its documented Linux and SQLite execution path is distinct from the unverified DSQL deployment path. The HQ profile is a migration candidate with explicit preservation requirements, rather than evidence that the live fleet has migrated. [S1-S3]

The inspected v4 implementation is the AWS Micro v4 cell under `customer/Kainersdorfer Wolfgang/wolf-sola`. It exposes identity, iteration, history, voice and calendar functionality and supports runner secret references. Its README describes a single live deployment with stage pointers reserved for future separation. This draft does not infer isolation from the presence of those pointers. The checkout contains several other lineages; the label v4 does not uniquely identify them. [S4]

The existing `data-custody.v1` standard distinguishes subject, custodian and requester, and custody levels L1 through L4. Those levels describe custody mechanisms. They MUST remain distinct from the composition assurance levels defined here. [S5]

Classification of every artifact

An artifact is an identified value or immutable version of an object. It MUST carry, directly or through an authenticated manifest, its identity, content digest where meaningful, kind, schema, provenance and label. Mutable objects MUST have version identities; a proof about one version cannot silently apply to another.

The mandatory security label is $L = (A, T)$:

- A is the audience: the principals authorized to receive the artifact. Public is the universe of principals; an empty audience permits no delivery.
- T is its integrity or trust classification in a declared lattice. For a simple profile, `untrusted <= reviewed <= verified`. These names designate policy categories, not automatic guarantees about truth or correctness.

Groups MUST resolve to authenticated membership under a recorded policy epoch. A deployment MUST specify how expiry and membership revocation affect stored values and active grants. A group name without resolution semantics is insufficient for a proof.

Additional classification fields include subject, custodian, purpose, retention, residency, provenance and applicable domain rules. These constraints MUST be checked independently unless the profile supplies a sound algebra for them. For example, combining two residency requirements may produce an impossible placement; the system must reject it rather than discard one.

Code and bricks use exactly the same A and T dimensions as data. Code is both readable information and a potential source of behavior. Its execution rights are therefore recorded separately from its label. A signed binary is not authorized merely because it is signed; its signature establishes provenance, subject to the signing key's trust.

Artifact	Example label	Additional obligations
Public schema	Public, reviewed	Version and compatibility contract
Owner record	Owner and delegated readers, untrusted	Custodian, purpose and expiry
Brick source	Maintainers, reviewed	Dependencies and execution contract
Generated patch	Authorized source readers, untrusted	Base revision and verification receipt
Installed binary	Authorized operators, verified	Build provenance and bounded authority
Security proof	Authorized reviewers, verified	Theorem, assumptions and checker version

The labels in this table are examples, not universal defaults. Unknown audience, provenance or required contract MUST cause quarantine or denial. An implementation MUST NOT treat missing metadata as public or verified.

Flow algebra

Define $L1 \leq L2$ to mean that information may flow from L1 into L2 without an exception:

$A2 \text{ subseq } A1$ and $T2 \leq T1$.

Thus a destination cannot have extra readers, and combining information cannot raise its integrity. The conservative combination of inputs is:

$L1 \text{ join } L2 = (A1 \text{ intersection } A2, T1 \text{ meet } T2)$.

This operation is associative, commutative and idempotent when audiences are sets and trust is a lattice. It gives the same classification whether a system combines artifacts directly or combines already classified intermediate results. These algebraic properties do not themselves prove execution safety.

A computation's output MUST include the labels of all inputs that can influence it, including code, configuration, control decisions and relevant error paths. If an implementation cannot establish a more precise dependency set, it MUST use all inputs visible to that execution context. Installing reviewed code into a computation that reads untrusted records does not make its result reviewed.

An agent trajectory MUST accumulate the classifications of material admitted to its context. Ordinary reads can only restrict that trajectory. Effects are a separate durable history; approval is permission for a bound action. This adopts OpenAPPA's separation of audience, trust, effects and attention. [O1]

Inference providers, embeddings, telemetry, final responses, repository publishing and build services are destinations. Their operators and configured retention policies MUST be included in the recipient model. A tool-only guard cannot justify a claim covering model requests or unrestricted subprocess networking.

LLM deployment and security profiles

LLM hosting arrangements MUST be classified separately from artifact labels, custody levels and proof assurance. An API, rented GPU, owned machine or on-premises deployment describes an operating arrangement; none establishes a security rank by itself. Each arrangement can cover overlapping ranges of security outcomes. The comparison depends on the protected assets, adversary, operational controls and available evidence.

Ownership, location and operation are independent attributes. An owned machine can sit in a third-party datacenter; an on-premises appliance can be vendor managed and send telemetry externally. A manifest MUST state the inference operator, hardware owner, physical location, tenancy, administrative access and connected services instead of inferring them from a marketing label.

Arrangement	Main allocation of responsibility	What must be established
Managed inference API	Service operator manages inference infrastructure; customer manages access and integration	Actual recipients, retention, routing, account controls and service evidence
Rented compute	Customer manages the model and serving stack; infrastructure operator controls underlying facilities	Tenant isolation, host access, image integrity, patching and storage handling
Owned compute	Owner controls hardware procurement; operators and location may vary	Administration, firmware, physical custody, maintenance and supply chain
On premises	Processing occurs at a declared customer site; ownership and management may vary	Local access, egress, logging, updates, remote support and operational competence

Every inference brick MUST publish a security profile covering these dimensions:

- **Exposure:** who can access prompts, responses, embeddings, caches, logs, backups and model adaptations, including administrators and support personnel.
- **Persistence:** retention, deletion, training use, abuse review, crash dumps and residual storage or accelerator memory.
- **Isolation:** tenancy boundaries, process and network confinement, workload identity and separation of management authority.
- **Operational protection:** patching, credential handling, monitoring, incident response, physical security and recovery.
- **Control and sovereignty:** operator authority, approved locations, jurisdictional dependencies and the ability to operate without external services.
- **Integrity and evidence:** provenance of weights and serving code, configuration verification, attestation where available, independent assessments and proof scope.

Availability MUST also be assessed where it matters to the use case. Strong confidentiality does not establish continuity, and redundant infrastructure does not compensate for an unauthorized recipient.

Security levels within each dimension

This draft proposes the following control maturity levels, applied separately to each dimension. They describe supported controls, not a probability of breach. The composition assurance levels A0-A5 continue to describe validation and proof evidence for the system.

Level	Meaning
C0 Unknown or unmet	Required facts are absent, or a required control is known to fail
C1 Declared	A named operator claims the control and assigns responsibility
C2 Enforced	The control is configured and its operation has been verified
C3 Independently assessed	Independent assessment or adversarial testing supports the control within a stated scope
C4 Formally supported	A checked argument supports the specific control, with implementation linkage and explicit assumptions

Assessments MUST distinguish unknown from known failure even though both prevent satisfying a requirement. A C4 result for a protocol MUST NOT promote physical security, operator conduct or an entire API to C4. Each rating MUST identify its evidence, assessment date, validity period and exclusions.

A deployment has a vector of dimension ratings. If uncertainty remains, each component MAY be expressed as a justified range [lower, upper]; for example, [C1, C3] means C1 is established and C3 is a plausible ceiling pending specified assessment. It does not assert a statistical confidence interval or authorize treatment as C3. Missing facts MUST NOT be filled with favorable assumptions. A range alone cannot describe whether a control meets the use case, so the actual control properties MUST accompany it.

No universal average security score is defined. For the same threat model, one profile dominates another only if it satisfies at least the same required properties in every relevant dimension and improves at least one. Otherwise the profiles are incomparable. Different weaknesses can produce overlapping ranges even when one arrangement offers more direct control.

Managed API and on premises comparison

Consider two hypothetical deployments: a managed API with restricted identities, verified private connectivity and acceptable model-specific data handling; and a local GPU server with a shared administrator account, stale software, exposed serving ports and prompt logs readable by many staff. The API can provide stronger protection against those operational threats. This is a conditional design example, not a measured ranking of AWS against a particular local installation.

If the requirement is that no external operator may process plaintext, the managed API can instead be ineligible regardless of its operational strength. A well-administered local deployment with controlled egress may satisfy that requirement. The standard therefore compares eligibility against explicit constraints before comparing maturity. Ownership alone does not establish eligibility.

Amazon Bedrock illustrates why assessment must identify the exact service configuration. AWS documents infrastructure and customer responsibilities, separation from model-provider access, and support for private VPC connectivity. Its current retention documentation distinguishes modes and model requirements; a generic claim that every Bedrock request has zero retention is insufficient. The applicable endpoint, model, Region and effective settings must be checked. [B1-B3]

Inference admission and composed proofs

Before a request leaves its execution context, the monitor **MUST** check that the inference profile satisfies the artifact's audience, purpose, retention and location requirements. The recipient model **MUST** include entities that can observe plaintext under the profile's threat model. Transport encryption and encryption at rest do not themselves exclude a serving operator from plaintext processing.

Each inference manifest **MUST** bind the model revision, serving operator, endpoint or machine identity, effective policy, route, logging and retention settings, relevant evidence and assessment expiry. The contract **MUST** classify prompts, retrieved context, responses, caches and adaptations. Model responses **MUST NOT** acquire execution authority merely because inference ran on trusted infrastructure.

Routers, fallback models, remote embeddings, external safety services, telemetry and support tunnels are separate composition paths. Every permitted alternative **MUST** meet the request's mandatory constraints. A local-first profile that silently falls back to an external API is nonconforming when external inference is prohibited. Fallback changes require admission checks before disclosure.

The composite profile **MUST** preserve every relevant path's requirements and evidence. Weakest-control reasoning applies only to the same property on a path that actually influences it; a numerical minimum across unrelated ratings cannot establish security. A5 proofs **MUST** include inference routing, prompt disclosure and resulting effects within their stated scope. A managed service **MAY** be an explicitly assumed boundary; such a theorem **MUST** state that provider internals were assumed rather than proved. Local hardware has an analogous boundary for firmware, drivers and privileged operators.

Brick contracts

A brick is a reusable capability whose identity binds source, executable artifacts, interfaces, dependencies and security contract. Its manifest **MUST** declare:

1. Artifact identity, version, provenance, classification and dependency digests.
2. Input and output schemas, label transfer rules and preconditions.
3. Required capabilities, accepted issuer identities and principal scope.
4. Permitted effects, destinations, filesystem roots, network routes and secret references.
5. Persistent state ownership, retention and uninstall behavior.
6. Isolation assumptions, failure behavior and resource bounds.
7. Referenced standards and available evidence, including known limitations.

Requirements **MUST** describe behavior rather than a preferred language or vendor. The cell authenticates and coordinates; adapters translate protocols; policy enforcement decides whether an operation is permitted. Business components **SHOULD** reference shared standards while keeping their implementation contracts next to their code, following the v5 library's documented organization. [S1]

A reusable brick package **MUST** contain no owner's private state or reusable credential. A deployed instance **MAY** receive narrowly scoped credentials through a private provisioner. Capability authority **MUST** be granted explicitly; combining bricks **MUST NOT** union their permissions into a shared privileged identity.

Recursive compositions

A composition is an identified graph of brick instances, data stores, principals, destinations and mediated channels. Edges describe allowed communication and delegated authority. The

graph MUST include infrastructure that can observe or affect protected values, including proxies, databases, queues, providers and factor adapters.

For each connected output and input, a validator MUST establish schema compatibility, legal label flow, adequate capability scope, compatible policy constraints and satisfied isolation assumptions. It MUST also check indirect paths and shared resources. Pairwise interface checks alone cannot exclude a path through a log, shared directory or confused deputy.

A composition MAY be packaged as a brick and composed again. Its external contract MUST soundly summarize everything visible at its boundary, including outputs, failures, timing where in scope, effects and authority. Internal dependencies remain in its locked manifest. A summary MUST overapproximate possible behavior; it cannot hide a weaker internal guarantee behind a stronger external declaration.

The composite artifact's own label describes who may read its packaged contents and their integrity. Its runtime contract describes transformations of instance data. These are distinct: packaging public code with a private configuration makes that package private, while running public code on private records makes the resulting values private without changing the public source artifact.

The baseline composite inherits the most restrictive compatible constraints. Its permissions are explicit grants at each boundary. Its assurance is supported by composition evidence, rather than a numerical average or the minimum of component badges. Cyclic graphs require an invariant or fixed point argument; a topological pass is insufficient.

Enforcement and policy governance

A reference monitor MUST operate outside the agent's editable authority and mediate every operation covered by the claim. It MUST bind the decision to the actual arguments, artifact versions, recipients and policy digest used at execution. Changes between authorization and execution require reevaluation or an atomic reservation.

Policies SHOULD be layered by organization, domain, cell and instance. A refinement MUST preserve mandatory parent restrictions, or carry an explicitly authorized exception. The validator MUST reject requirements the deployment cannot enforce. This extends OpenAPPA's documented approach to configuration layering into a SOLA deployment requirement. [O2]

Contract resolution MUST be deterministic and versioned. Ambiguous selectors, invalid matched schemas and unknown capabilities MUST fail closed. If an OpenAPPA adapter is used, its mapping MUST preserve the selected engine version's contract semantics; this draft does not supply a new `appa.toml` schema.

Label transitions, policy decisions and effects MUST survive restart and context compaction. Concurrent branches MUST share a causally ordered history for global restrictions. A crash during an external operation MUST record an uncertain effect until reconciliation establishes the outcome; uncertainty must not become an automatic retry of a destructive operation.

The runtime MUST protect against alternate access through symlinks, hard links, environment variables, subprocesses, inherited descriptors and caches wherever those paths are covered by the profile. File annotations without confinement are inadequate for arbitrary code execution. OpenAPPA's file runtime documents a limited mediated scope and separate isolated processing; SOLA must make its own coverage explicit. [O3]

Controlled exceptions

Declassification changes the audience of a particular output. Endorsement raises the integrity assigned to a particular artifact. They are different privileged operations, and neither globally clears an agent's accumulated label.

An exception MUST identify the authorized actor, exact input and output digests where available, transformation, recipient set, purpose, expiry, policy version and affected obligation. Approval MUST be bound to the exact action and consumed according to the contract. Login approval MUST NOT be reused as permission to deploy or disclose records.

A sanitizer MUST have an explicit output contract and independent evidence for its claimed transformation. Removing names does not prove anonymity. A reviewed change MAY receive an endorsement for a named invariant after protected verification; passing tests does not endorse every security property of that change.

Separate execution contexts MAY preserve utility by handling sensitive inputs independently. Their return channel is itself a mediated edge. Creating a new context MUST NOT allow restricted state, files or credentials to cross without classification.

Assurance levels

These proposed levels are cumulative. The prefix A distinguishes them from custody levels L1-L4.

Level	Required evidence
A0 Declared	Complete manifests, labels and claims; no security assurance
A1 Validated	Deterministic static contract and composition validation
A2 Enforced	Runtime mediation and deployment coverage evidence
A3 Tested	Independent adversarial, failure and concurrency checks for the exact deployment
A4 Component proved	Machine checked component properties under stated assumptions, plus A3
A5 Composition proved	Machine checked end-to-end properties for the locked recursive graph, plus implementation refinement and inherited obligations

These are requirements for future assessment. This document does not assign an existing brick any level. A4 proofs for every brick do not establish A5. An unproved adapter can invalidate the relevant end-to-end theorem even if all application bricks have proofs.

A5 claims MUST identify the property, observer model, permitted exceptions, proof scope and trusted computing base. They MUST distinguish confidentiality, integrity, authorization, effect ordering and availability. The phrase formally secure without those qualifiers is nonconforming.

Proof obligations for combinations

For each brick i , define an assumption P_i , transition model M_i and guarantee Q_i . A composition proof MUST show that the environment and connected components establish every P_i , and that the combined transitions preserve a global invariant I . In symbolic form:

$\text{Initial}(s) \text{ implies } I(s).$

$I(s) \text{ and Step_C}(s, s') \text{ implies } I(s').$

The invariant MUST include valid label transitions, capability confinement, policy integrity, grant scope, mediated destinations and the declared effect ordering. For cyclic interactions, the proof MUST establish mutually compatible assumptions without circularly assuming the desired guarantee.

Confidentiality additionally requires an observation property. For each unauthorized observer u , executions differing only in data u may not read must produce equivalent observations for u , except for specified releases. A profile MUST state whether this equivalence is termination insensitive or includes timing, scheduling, traffic volume and failure behavior. Label preservation alone is not a full noninterference proof.

Integrity requires that low-trust inputs cannot cause a protected high-trust decision unless a named endorsement rule permits that influence. Authorization requires that each effect is exercised under the correct scoped capability. Effects such as sending and deploying require state-machine obligations for approval, idempotency and uncertain outcomes.

An implementation refinement argument MUST connect source and deployment behavior to Mi: unmodeled I/O, mismatched dependencies or an editable monitor breaks that connection. Cryptographic proofs MUST name primitive assumptions. OS, runtime, compiler, database, policy authorities and membership services belong in the trusted computing base whenever the theorem relies on them.

Every proof package MUST contain the locked composition and policy digests, theorem statements, assumptions, executable checker instructions, proof dependencies, checker version and verification receipt. A receipt MUST state which theorem passed; a successful process exit without identified obligations is insufficient.

Changes to code, contract, policy, membership semantics, dependency, build toolchain or deployment invalidate affected claims until impact analysis and required rechecking succeed. A proof MAY be reused only when the applicability of its assumptions to the new instance is itself established.

Worked combination

Consider a profile containing an owner record store, a coding runner, a verifier and a publisher. The store permits the owner to read a customer record. The runner receives a private prompt and public source; the customer record is untrusted input. Its patch and logs inherit the private audience and lower trust. The verifier can establish a particular code property but cannot make the patch public simply by returning success.

Publishing the patch to a public repository is denied because public readers exceed the patch audience. An authorized transformation may produce a separate candidate that excludes protected influences, or an authority may approve a specific release within its powers. Either path must create evidence bound to that candidate. Sending the private prompt to an external model is a separate destination decision made before inference.

Deploying the candidate is a separate effect requiring the selected deployment capability and applicable review. A5 for this graph would require proof of classification propagation, protected verification, release scope and deployment authorization, together with evidence that agents cannot bypass the channels through arbitrary networking or production mounts.

For the v5 authentication composition, the factor adapter may attest a challenge but must not read owner records or redeem the challenge. Composing the login UI, Matrix factor and backend must preserve this separation and bind approval to the exact challenge. The homeserver and adapter remain explicit trust assumptions. [S2, S6]

Automatic documentation from Java contracts

The Java reference profile adopts a source-driven documentation workflow. Adam Bien's SBCE places component requirements in package documentation, while his ebank example uses a component-local Requirement annotation with enum IDs to link implementations and tests. His jmarkdoc doclet exports JavaDoc to Markdown and renders requirement documentation tags.

These are complementary mechanisms: the Java annotation is compiler-resolved metadata; the lowercase JavaDoc tag is documentation markup. [J1-J3]

The authoritative component contract **MUST** remain in `package-info.java`. It defines the requirement statements and stable IDs. A generated `Requirement.java` vocabulary supplies one enum constant per statement. Methods and tests reference those constants without copying the requirement prose. IDs **MUST** be qualified by component identity so that identity R1.1 and iteration R1.1 remain distinct.

For example, a canonical package statement can describe identity R6.5 as allowing at most one grant when redemptions race. The redemption method then carries this reference:

```
@Requirement({R6_1, R6_2, R6_3, R6_5})
public Response redeem(/* transport arguments */) {
    // Implementation of the component operation.
}
```

The existing concurrent-redemption test references R6_5. The compiler rejects a nonexistent enum ID. A generator joins the resolved references to the package statements and produces readable contracts, API documentation and a structured trace report. This example is schematic; the working v5 annotations have been applied to the actual method and tests.

Reference generation pipeline

The initial implementation is `sola-master/library/tools/contract-docs`. It extracts canonical statement IDs, checks the adopted enum vocabularies, builds a pinned jmarkdown source revision, renders API documentation and uses a Java annotation processor to export resolved requirement links. It then generates Markdown and JSON reports with source hashes and optionally derives a PDF through Pandoc and XeLaTeX.

The processor runs through `javac` with the project's Maven test dependency classpath. Unknown symbols and orphan requirement references fail generation. Missing links remain visible as gaps. Publication **MUST** use a fresh output destination so that a failed run cannot masquerade as an updated report. Normal generation does not rewrite the source vocabulary; synchronization is an explicit development command.

The first adoption covers v5 identity's nine existing requirements, linked to five HTTP boundary methods and nine existing test methods. Other component statements are documented but remain visibly unlinked in the annotation report. This bounded adoption adds no runtime policy enforcement. The annotation has `SOURCE` retention because this pipeline needs compile-time metadata rather than runtime reflection. This differs from ebank's runtime-retained annotation by deliberate design.

Jmarkdown's source-only rendering fallback is useful for documentation but **MUST NOT** be presented as successful compilation. The separate annotation-processing step supplies resolved references. Project test execution remains a separate verification step; generated trace metadata records that distinction.

Security metadata and proof evidence

Requirement links establish declared traceability. They **MUST NOT** promote an artifact's trust, a control maturity rating or a composition assurance level. A linked test may have weak assertions, and a test method annotation does not establish that every parameterized row tests its stated requirement.

Future security annotations **MAY** reference a standard, artifact classification, brick contract, inference profile or proof obligation by stable ID. The referenced definitions **MUST** be versioned and generated into machine-readable manifests. Deployment measurements, approval receipts and

proof-checker outputs remain external evidence bound to digests. A source annotation such as a declared on-premises profile cannot establish that the running system has no external telemetry.

CI SHOULD regenerate projections, reject stale vocabularies and orphan links, run behavioral tests, and verify applicable deployment or proof evidence before publishing assurance claims. Generated documents MUST identify the source versions and distinguish declared, tested and proved properties. This preserves language-independent commons standards while making Java implementations explainable and reviewable from their own source.

Conformance and adoption

The first deliverable is a language-independent schema for artifact, brick, composition and proof manifests, followed by a deterministic validator. The next deliverable is a runtime enforcement profile for one confined Linux deployment. A minimal record-read to publish example should supply both a permitted case and a rejected leakage path before extending to the full HQ composition.

The conformance suite MUST exercise indirect and recursive flows, untrusted input affecting protected operations, provider egress, secret-bearing logs, unknown labels, contract mismatch, replayed approvals, membership changes, crash recovery, concurrent effects, filesystem aliases, dependency substitution and policy weakening. Inference checks MUST also cover prohibited fallback, stale assessments, changed retention or routing settings, inaccessible evidence and on-premises telemetry escaping the declared boundary. Proof checkers MUST reject a deliberately broken example. Tests establish observed behavior within their scope; they do not replace A5 obligations.

Legacy v4 cells MAY conform through adapters that supply the required manifests and enforcement. Their legacy authentication contract remains explicit. V5 SHOULD be the first reference target because its library already separates optional bricks and deployment from candidate verification. This is a design choice, not a claim of current conformance.

Before adoption, the working group must settle audience membership semantics, the trust lattice, supported side-channel scope, proof tooling and policy exception authority. The core recommendation is stable across those choices: classify every artifact, constrain every capability, validate every connection and prove the actual composed system at the highest level.

Sources and evidence boundaries

Local sources were inspected on 2 October 2026 at sola-master HEAD 8f052809. Working-tree content may differ from that commit. The implementation review was a targeted source and contract inspection; it did not rerun deployments or certify the fleet.

- S1: sola-master/library/README.md and library/bricks/README.md.
- S2: library/cells/sola-zero-v5/README.md and identity controller Access.java.
- S3: library/profiles/hq-v5.md.
- S4: AWS Micro v4 customer README; iteration HTTP adapter IterationResource.java in the same cell.
- S5: sola-commons/standards/data-custody.v1/SPEC.md. Its custody levels are not proof certification levels.
- S6: library/standards/key-and-factor-v1/SPEC.md and library/bricks/verified-command-runner/README.md.
- O1: [OpenAPPA How it works](#), accessed 2 October 2026. Supports the attributed trajectory and contract concepts; the SOLA recursive proof system is proposed here.
- O2: [OpenAPPA Basic Principles](#). Search-index excerpt retrieved 2 October 2026; the full page returned 404. Used only for the attributed stacking and deployment validation ideas, pending confirmation against a maintained source.

- O3: [OpenAPPA Coding agents](#), accessed 2 October 2026. Documents file mediation and its stated boundaries; it does not establish arbitrary end-to-end SOLA security.
- B1: [Amazon Bedrock Data protection](#), accessed 2 October 2026. Supports the attributed responsibility and model-provider access descriptions.
- B2: [Amazon Bedrock Data retention](#), accessed 2 October 2026. Settings and model requirements must be checked for the actual deployment; this document makes no blanket zero-retention claim.
- B3: [Amazon Bedrock PrivateLink](#), accessed 2 October 2026. Private connectivity does not itself establish all confidentiality or integrity properties.
- J1: [Adam Bien SBCE](#), accessed 2 October 2026. Supports package documentation as the component contract.
- J2: [Adam Bien ebank Requirement annotation](#), inspected at the named revision. Supports the component-local enum annotation pattern.
- J3: [Adam Bien jmarkdoc](#), inspected and built at the named revision. Supports Markdown documentation generation and requirement documentation tags. The SOLA annotation trace processor is an addition made for this draft.

OpenAPPA marketing percentages and benchmark results are not used as evidence for this draft's proof requirements. The algebra, recursive contracts, assurance levels, LLM profile dimensions, control maturity levels and adoption requirements are original proposals for review. No LLM deployment was measured or assigned a rating in preparing this revision.