GOVERNED ACTION INTERCHANGE PROFILE

GAIP Candidate Specification

A profile for exchanging and verifying a governed action

Candidate — Open Review. No conformance scheme is defined.

Version	Version 0.4
Status	Candidate — Open Review
Date	26 August 2026
Author	Yves Frédéric N'Soussoula, Founder & CEO
Publisher	GoSec Cloud OÜ, Tallinn, Estonia
Licence	Creative Commons Attribution 4.0 International
Web	www.vianordis.eu

Open Review. This document is published for criticism. Corrections of fact are applied in the next edition and listed; substantive critique is credited unless the contributor prefers otherwise. Contact: www.vianordis.eu/en/contact · Law as at 26 August 2026. Not legal advice.

Contents

1. Status and purpose

1.1 What this document is

GAIP is a candidate profile. It defines the minimum contract between the components that decide, approve, execute and evidence an action taken by an AI agent or comparable automated actor on behalf of a principal.

It has three goals:

1. make the object approved by a human the same object evaluated and constrained by the enforcement point;
2. carry parameter provenance and material target-state preconditions with that object;
3. produce a verifiable relationship between the authority decision and the committed result.

GAIP does not define an AI model interface, a policy language, an identity provider, a workflow engine or a target-system API. It defines the contract between them.

The problem this profile addresses, the regulatory context and the limits of the whole approach are set out in *Vianordis Position Paper 01 — The AI Control Gap*, v0.5.4. Readers who have not read §3.2 of that paper — on why a perfectly bound approval can still authorise a fraudulent payment — should read it before implementing anything here.

1.2 What this document is not

Not a standard. This is not an IETF, ISO, CEN/CENELEC, ETSI or regulatory standard, and it is not on a track toward becoming one at the time of writing.

Not vendor-neutral, yet. GAIP is designed to be implementable without reference to any particular product, and nothing in it depends on Vianordis. But it is authored by a single vendor, published by that vendor, and has no independent implementations, no working group and no public repository. Vendor neutrality is an objective for this work, not a property it currently possesses. It becomes true when someone else implements it and disagrees with us in public.

Not a conformance scheme. Version 0.1 defined five named levels — GAIP-C, GAIP-S, GAIP-E, GAIP-H and GAIP-X. **They are withdrawn.** A five-level scheme with no test suite, no test vectors, no attestation format, no validity period, no revocation, no assessor role, and no separation between product conformance and deployment conformance produces exactly one outcome: self-assertion that nobody can refute. The first such assertion would in all likelihood have been ours. Version 0.1 also required implementations to publish test vectors before claiming interoperability, while publishing none itself.

This edition therefore makes no conformance claim available. §23 publishes the first test vectors. A conformance scheme can be built when there is a test suite, an attestation format and at least one implementation that is not ours. Until then, an implementation may state which requirements of this document it meets, for which action types, at which date — and that statement is a description, not a certification.

Not yet implementable interoperably. This edition is not sufficient for independent interoperable implementation, because its normative schema and its transport binding remain unpublished. Two careful implementations of this text can still produce structurally incompatible envelopes.

Not complete. §26 lists what is still open. Several items there are load-bearing.

1.3 What changed from 0.1

Version 0.1 was reviewed adversarially before publication and nine blocking defects were found. This edition addresses them:

0.1 defect	Where addressed
Provenance class declared by the component it constrains	§7.3 and §7.4 — resolver attestation, deterministic derivation, unattested claims fail closed to P4
Action tier declared by the requester	§7.2 and §7.5 — tier derived by the PDP from a signed, revocable action-type registry, recomputed and re-checked at the PEP
Approval not bound to what the human saw	§18 — normative presentation profile; `presentation_digest` defined in §10.3 and carried in the approval record at §9.7

0.1 defect	Where addressed
Digests undefined and confusable across domains	§10 — input scope, label prefixes, inline algorithm, test vectors
No threat model	§4
Signer roles open; composite signature collapses onto one key	§11 — normative signer matrix; inner signatures preserved
Canonicalisation before signature verification	§12.6 — verify over the octets as received; reject non-canonical input
Break-glass named but never specified	§19
No schema, media types or wire binding	§22

Beyond those: cumulative mandate budgets (§17.1), delegation chains (§17.2), four-eyes and quorum (§18.3), write-ahead evidence (§20.2), a broker-ledger assurance mode for targets without conditional writes (§15.3), an `AwaitingReceipt` state for asynchronous receipts (§14), completion of the state machine, and monetary amounts and large identifiers carried as strings rather than JSON numbers (§12.3).

Four invariants changed in the position paper and are carried here: **GI-3** is now declare-and-detect rather than an absolute no-bypass claim; **GI-8** now constrains the enforcement point, not only the benefiting application; **GI-1** runs through the full delegation chain; and **GI-5** binds to the rendered presentation.

1.4 What changed in 0.2.1

An external validation review of the 0.2 set found twelve further defects. All are addressed here, and all of them tighten a requirement or withdraw a claim:

Finding	Where addressed
`policy_decision_digest` mandatory but undefined	§10.2 — label `GAIP-0.4/policy-decision/v1` with an exact input scope
The Action Type Registry controls every tier-indexed rule and had no signer, no lifecycle and no change control	§7.5 and §11.1 — Registry Authority, signed revocable snapshots, hash-chained versions, commit-time revocation check, independent approval for tier-lowering changes
The presentation digest proves what the service rendered, not what the human saw	§18.1.1 — two declared positions, trusted rendering path or trusted approval client, with the weaker one named as a trust assumption
`broker-ledger` was treated as if it satisfied pre-commit freshness	§15.1 and §15.2 — declared detective-only, explicitly not equivalent to a conditional or locked commit
The idempotency key prevents envelope replay, not business duplication	§13.1.1 — a separate business deduplication key declared per action type, with its own error class
Releasing an indeterminate budget reservation on a timeout permits silent overspend	§17.1 — reserved / committed / released / **quarantined**; a quarantined reservation is released only by authoritative reconciliation
Safe degradation appeared to permit a tier 3 side effect while evidence was down	§21.3 — queue or block; write-ahead evidence is a precondition of execution from tier 3
No deterministic mapping from attestation evidence to P0–P4	§7.4 — a normative derivation table with precedence rules
`value_ref` could bind a mutable locator rather than a value	§9.4 — content-addressed, immutable, with a commitment to the resolved plaintext
No array ordering rule, though JCS preserves order	§12.4 — semantic and non-semantic arrays, with sort keys for the latter
The string rule for numbers collided with structural integers	§12.3 — exact decimals and unsafe identifiers as strings; bounded structural integers as JSON integers
JWS General Serialization does not preserve inner signatures	§8 — composites embed complete inner objects byte-for-byte

Prior art was also corrected. **RFC 9396** and **PSD2 dynamic linking** were missing and are now in §3, and the "three defensible contributions" framing in §3.1 is replaced by a composition claim that asserts no priority, plus an explicitly disputable list of four properties.

1.5 What changed in 0.2.2

A second validation pass found that 0.2.1 had closed its twelve findings textually while leaving eleven normative gaps, several of them security-relevant. This revision closes them.

Finding	Where addressed
0.2 and 0.2.1 were **indistinguishable on the wire**, so a policy could not require the revised semantics and downgrade protection could not see the difference	§9.1.1 — `profile` carries the exact revision, is mandatory in `critical`, and is inside the approved-action digest scope
`registry_digest` and `previous_version_digest` were mandatory with **no digest definition**	§10.2 — six new labels, including `GAIP-0.4/action-type-registry/v1`, with a published registry test vector (§23, TV-6)
The approval signature did **not cover** `approver_role`, `auth_assurance_level` or `session_binding`, though quorum and segregation decisions rely on them	§10.3
The approved-action digest bound each parameter's **value and class but not its attestation**, so a different attestation with the same value could be substituted	§10.2, §9.4
The approved-action digest omitted `allowed_adapter` and `assurance_mode`, so a downgrade from conditional commit to broker-ledger did not break the binding	§10.2
Missing low-tier provenance defaulted to P2, an authenticated assertion, on the strength of no evidence at all	§7.4 — absence never yields P2. (0.2.2 introduced a class **PX** for this; 0.3 replaces it with an assessment status — see §1.6)
Provenance derivation named singular fields while P0 requires two resolutions and a comparison	§7.3, §7.4 — a `resolutions[]` array, an explicit `comparison`, an `independence_basis`, and authoritative sources declared per parameter in the registry
"At least one precondition" at tier 4/5 could be satisfied by an **irrelevant** one; the "profile default" freshness window did not exist	§7.5, §9.6 — `required_preconditions` declared in the registry, an explicit `no_material_preconditions` declaration where none exist, and `freshness_window` mandatory from tier 3
The envelope carried `issued_at`, `not_before` and `expires_at` and **nothing required a verifier to check them**	§13 step 5
Write-ahead evidence was **required of the very component the threat model assumes compromised**	§20.2 — the Evidence Service issues a one-time execution proof after checkpointing the commit intent, and an unresolved intent raises an anomaly with no sequence gap
The business deduplication key was checked but not **reserved**, so two concurrent envelopes could both pass	§13.1.1

Two invariants were also split in the position paper and are carried here: **GI-5** into **GI-5A** (approval artefact equivalence) and **GI-5H** (human-view equivalence), and **GI-6** into **GI-6P** (preventive state freshness) and **GI-6D** (detective state divergence). The reason is stated in §15.2 and §18.1.1: this profile permits modes that cannot deliver the strong property, and describing them as a "detective form" of the same invariant concealed a real loss of assurance.

Wire compatibility. `profile` carries the exact revision and is inside the approved-action digest input, so it cannot be stripped.

1.6 What changed in 0.3 — and why this is not 0.2.3

A third validation pass found two missing cryptographic bindings, an unimplementable registry rule, incomplete reservation and execution-proof semantics, and an inconsistent digest wire representation. Correcting them **changes signed scopes and wire artefacts**, so this is a minor-version revision rather than a patch, and envelopes are not interchangeable with 0.2.x.

Finding	Where addressed
The attested value was never bound to the action parameter. An attestation carried a `value_digest`; the parameter carried a value and an `attestation_digest`; nothing required the verifier to prove they described the same value. A valid P1 attestation for one value could be presented alongside a different value in the parameter, and the class would be applied to the wrong value	§7.3, §7.4, §10.2 — a canonical typed **parameter-value commitment**, recomputed by the verifier and compared against every resolution's `value_digest` before tier derivation and before the matrix
The approval did not bind the structured intent. The digest covered `intent{type, schema}` only, so two materially different intents sharing a type and a schema URI produced the same digest, and a mutable URI was the only schema commitment	§9.3, §10.2 — an `intent_digest` over type, schema URI, **schema digest**, values, origin and the principal's confirmation record
The registry could not express its own threshold model. Two entries for the same action type and target at different thresholds were rejected by the §12.4 sort rule, and no operator vocabulary, precedence rule or default tier existed	§7.5 — an immutable `entry_id`, a canonical typed condition language, a most-specific-match algorithm and a mandatory default entry
Budget reservation was not required to be atomic, and no failure released either reservation. Two concurrent transactions could both observe sufficient budget; a failure at steps 9 to 14 leaked both reservations	§17.1, §13.3 — atomic compare-and-reserve, and a mandatory rollback step

Finding	Where addressed
The execution proof was load-bearing and had no artefact. No data model, no digest domain, no signer, no issuer or audience rule, no one-time consumption rule, no replay test	§20.2.3 — a signed `ExecutionProof`, its digest domain, the Evidence Service as signer, and four negative tests
The GI-6D reconciliation interval was declared but bound to nothing, so an operationally useless interval satisfied the requirement trivially	§9.1, §10.2, §18.1 — `reconciliation_interval` and `maximum_detection_latency` inside `execution_constraints`, inside the approved-action digest, shown to the approver, and capped by policy per tier
TV-6 violated the profile's own array-ordering rule, so the published registry vector was not a canonical registry	§23 — recomputed; and §12.4 now covers every array introduced since 0.2
Digests appeared in three representations — `{alg, value}`, multihash and `sha256:` strings — including inside the vectors	§10.1 — one representation, `{alg, value}`, used everywhere including the vectors
PX was a class in a taxonomy the other two documents do not have	§7.4 — withdrawn. Provenance assessment is now a **status**, not a degree of assurance: `provenance_assessment: "assessed" | "not-assessed"`, and `derived_class` is absent when not assessed. The canonical taxonomy stays P0–P4 across all three documents

Twelve further normative items from the same pass are addressed in place: authenticated-asserter identity for P2, verifier-recomputed comparison, a source-independence model for P0, attestation validity, typed precondition operands, a content-addressed presentation profile, ordering algorithms for profile and registry versions, canonical decimal normalisation, exact time inequalities, a domain-labelled idempotency key, an explicit client-and-display integrity assumption for GI-5H, and a defined binary embedding for composites.

1.7 What changed in 0.4

A fourth validation pass found that 0.3, for all its new bindings, still let a **signed denial proceed toward execution** — the processing procedure validated a policy decision's signature, digests, obligations, expiry and tier, and never required its effect to be `permit`. It also found that 0.3's own new step 2 resolved a requester-supplied schema URI **before** signature verification, contradicting §12.6 and creating an unauthenticated external-resolution surface. Both were introduced by earlier corrections; neither was in 0.1.

Finding	Where addressed
A signed `deny` could pass through processing. No step required `effect = permit`	§13 step 10 and T51 — the effect MUST be `permit`; `deny`, `defer`, `indeterminate`, unknown or absent fails closed with `E-POLICY`, and blocking obligations must be discharged and evidenced before execution
External schema resolution before signature verification	§13 — schema resolution and `intent_digest` recomputation moved to step 6, after signatures; step 2 is now local framing validation only, and T52 asserts that no external resolution occurs before step 5
Four load-bearing digests had no construction: `intent.schema_digest`, `presentation_profile_digest`, `auth_evidence_digest`, `credential_scope_digest`	§10.2 — four new labels, with TV-10 to TV-13
The approved-action input scope did not match TV-2, and had no projection for an unassessed parameter	§10.2 — the scope now carries `value_type`, `unit`, `normalisation_profile`, `provenance_assessment`, and makes `attestation_digest` and `derived_class` conditional on the assessment status
TV-7's commentary contradicted the normalisation rule	§23 — three cases: canonical input, non-canonical input normalised to the same commitment, and input rejected because normalisation would lose precision
Tier 5 permitted a one-of-one quorum, so "four-eyes" was expressible but not required	§18.3 — an automated tier 5 action MUST carry approvals from at least two distinct natural-person credentials in distinct roles, each distinct from requester, beneficiary and each other
The ExecutionProof proved that evidence existed, not that the credential scope was authorised	§20.2.3 — the issuer independently validates the envelope and the decision, derives the credential scope itself, and binds `credential_scope_digest`; the released credential MUST be one-time and transaction-bound
Currency and unit were unbound. An operand carried `unit: "EUR"`; the `amount` parameter carried none	§7.3.1, §7.5.1, §17.1 — parameters carry `unit`, conditions carry `unit_parameter`, the registry names `budget_amount_parameter` and `budget_currency_parameter`, and a currency mismatch fails closed

Nine further items are addressed in place: the registry independent-approval rule extended to removed preconditions, longer freshness windows, weakened authoritative sources, altered independence domains, weakened deduplication and broader adapter modes; equal-specificity registry matches now rejected rather than resolved by name; a signed `source_domains` mapping so a resolver cannot

invent independence; attestation required for any parameter capable of selecting tier 3 or above, before tier derivation; `maximum_attempts` and `evidence_profile_digest` bound into the approved action; stale step references corrected; the evidence record's "every parameter has a class" wording fixed; break-glass activation checkpointed before credential release; and the unevidenced market-wide claim about enterprise platforms removed.

---

2. Conformance language

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY** and **OPTIONAL** are used as requirement terms for this candidate profile.

They do not state legal obligations. A legal or regulatory requirement must be traced separately to its source.

Because no conformance scheme is defined (§1.2), these terms describe what an implementation must do **to be interoperable and to carry the assurance this profile claims** — not what it must do to earn a label.

3. Relationship to prior art

Nearly every mechanism here exists already, under a different name, in a standard or product that predates it. Stating that plainly is not a weakness in the argument; concealing it would be.

Capability in this profile	Existing standard or practice	What it already covers	What is outside its model
Policy Decision Point / Policy Enforcement Point	The **PDP/PEP** vocabulary originates in the IETF policy framework work — **RFC 2753** (2000) and **RFC 3198** (2001) — with antecedents in **ISO/IEC 10181-3:1996** (ADF/AEF). **XACML** v3.0 adopts that vocabulary and contributes **PIP** and **PAP**. **NIST SP 800-207** (August 2020) uses the same terms for zero trust	The decision/enforcement split, attribute-based evaluation	Purpose and monetary limits as first-class inputs; binding a decision to a *specific approved transaction*
Policy as code	**Open Policy Agent / Rego**, **AWS Cedar**	Versioned, testable, externalised authorisation logic	Domain-specific mandate semantics, tenant lifecycle, approval binding, evidence correlation
Mandate and delegation	**OAuth 2.0 Token Exchange (RFC 8693)** `act` / `may_act`; **macaroons** (Birgisson et al., 2014); **GNAP**; **UCAN**	Delegation semantics, actor chains, attenuable caveats, third-party discharge	Business purpose, cumulative risk ceiling, approval obligation and assignment-level revocation as attributes of the delegation itself
Transaction-specific authorisation	**OAuth 2.0 Rich Authorization Requests — RFC 9396** (Standards Track, May 2023). Its own payment example carries `instructedAmount {currency, amount}`, `creditorName` and `creditorAccount {iban}`	Fine-grained, per-transaction authorisation detail negotiated and carried in the authorisation request, and enforceable at the resource server	Parameter *provenance*; binding to a rendered presentation; mutable target-state preconditions; cumulative budget across transactions
Binding an authorisation to amount and payee	**PSD2 dynamic linking** — Commission Delegated Regulation (EU) 2018/389, Art. 5(1): the authentication code "is specific to the amount of the payment transaction and the payee agreed to by the payer", and "any change to the amount or the payee results in the invalidation of the authentication code generated"	Exactly the action-commitment property, mandatory in EU law since 2019, with an invalidation rule on change	Cross-system actions beyond payments; provenance of the bound values; precondition freshness; delegation chains; evidence correlation
Agent action governance at runtime	**Faramesh** (Fatmi, arXiv 2601.17744): an Action Authorization Boundary, a Canonical Action Representation, and provenance-complete decision records bound to frozen policy versions and state snapshots for deterministic replay. **AgentBound** (Kaul, Lan and Gupta, arXiv 2606.30970, June 2026): delegated authorisation, owner-signed behavioural constitutions, site action contracts, cryptographically signed **governance receipts** for post-hoc replay verification, and a standing-delegation model with per-execution policy refresh	Execution-time authorisation, canonical action encoding, policy-and-state-bound decision records, replay-verifiable receipts, refreshable standing delegation	Attested parameter provenance with a fail-closed class; rendered-presentation binding; cumulative monetary budget with reservation semantics; write-ahead evidence. Both are recent preprints, not adopted standards, and neither has been independently implemented either
Workload identity	**SPIFFE / SPIRE** — project started 2017, CNCF sandbox 2018, graduated 2022; mTLS; SCIM	Cryptographic workload identity, attestation, short-lived SVIDs, federation	Agent *instance* identity with parent-child lineage and principal attribution across a delegation chain
Session and entitlement revocation	**CAEP / Shared Signals**, OpenID Connect back-channel logout	Near-real-time session revocation across relying parties	Revocation of an in-flight *mandate* mid-process
Transparency services, signed statements and receipts	**SCITT — RFC 9943**, *An Architecture for Trustworthy and Transparent Digital Supply Chains* (Proposed Standard, IETF, 30 June 2026): COSE-based signed statements, registration policies, transparency services over append-only logs, receipts, and independently verifiable replay. Several Internet-Drafts — not adopted standards — now apply the architecture to AI-agent execution, pre-execution authorisation, action capsules and agent receipts	Registration, receipt issuance, append-only transparency and independent verification of signed statements, as a general architecture	The authority semantics: mandate, purpose, budget, provenance class, approval binding and precondition freshness. **This overlaps the evidence layer of §20 directly**, and a future edition should reuse SCITT rather than define a parallel transparency construction — see §26.2
Portable action envelopes and approval receipts	**PCAA** and **CAVA** (preprints, 2026): portable action envelopes, approval and runtime receipts, multi-checkpoint governance paths, canonical action identity and receipt integrity	Envelope portability, canonical action identity, approval binding, receipt integrity	Attested provenance with a fail-closed class; cumulative budget reservation; write-ahead evidence against the executing party. Preprints, not standards, and not independently implemented

Capability in this profile	Existing standard or practice	What it already covers	What is outside its model
Tamper-evident logs	**Certificate Transparency — RFC 6962** (CT v1, Experimental; formally obsoleted by **RFC 9162** CT v2, which is not yet the deployed profile), Trillian	Append-only logs, inclusion and consistency proofs, independent witnessing	Confidential enterprise evidence needs access control, selective disclosure, retention rules and private or federated witness operation
Supply-chain integrity	**SLSA**, **in-toto**, **Sigstore**, CycloneDX (Ecma-424), SPDX (ISO/IEC 5962:2021)	Provenance attestation, artefact signing, dependency transparency	Build policy, key custody, release authority, incident response
Canonical signed artefact	**RFC 8785** JSON Canonicalization Scheme (Informational, Independent Submission); **JWS (RFC 7515)**; **COSE** — RFC 9052 *Structures and Process* (STD 96) and RFC 9053 *Initial Algorithms* (Informational), together obsoleting RFC 8152; **RFC 9338** Countersignatures updates 9052; **RFC 9864** Fully-Specified Algorithms updates 9053	Deterministic representation and standard signature containers	The domain schema, authority semantics, provenance fields and processing state machine
Trusted time and long-term evidence	**RFC 3161**; **RFC 5816** (ESSCertIDv2 update, enabling migration off SHA-1 for signer-certificate identification); **RFC 4998** ERS; **RFC 6283** XMLERS; **RFC 9169** (new ASN.1 modules, Informational — it does not formally update RFC 4998)	Time-stamp tokens and archive evidence records	Operational trust model, key lifecycle, privacy, renewal policy, transaction correlation
State and concurrency binding	Conditional writes, optimistic concurrency, ETags and `If-Match` (**RFC 9110** §8.8.3 and §13), idempotency keys, saga patterns	Detecting stale state, preventing duplicate commit, handling partial failure	A cross-system rule for which preconditions bind to a human approval and carry into evidence
Management systems	**ISO/IEC 42001:2023**, **ISO/IEC 27001:2022** (Amd 1:2024), **NIST AI RMF 1.0** (NIST AI 100-1, January 2023)	Organisational governance, risk process, documentation, certification path	Runtime enforcement; per-transaction authority binding
Agent-to-tool integration	**Model Context Protocol (MCP)**	A standard transport and description format for tool access	Authority: MCP describes *how* to call a tool, not *whether this actor may*, for what purpose, under whose mandate

Note: an XACML 4.0 committee draft (CSD 01, February 2026) is in circulation; references above are to v3.0.

3.1 What is actually claimed as new

The claim is a **composition**, and it is deliberately stated at that level:

> GAIP is a candidate composition profile that combines attested parameter provenance, rendered-presentation binding, mutable target-state preconditions, full actor-chain attribution, cumulative delegation limits, write-ahead evidence, and cross-system decision-and-execution correlation. **It claims no priority over execution-time authorisation, transaction dynamic linking, or governance-receipt architectures.**

Version 0.1 and version 0.2 both listed three "defensible contributions". That framing is withdrawn. It does not survive contact with RFC 9396, which already carries transaction-specific authorisation detail; with PSD2 dynamic linking, which has bound an authorisation to amount and payee, with invalidation on change, in EU law since 2019; or with the 2026 agent-governance work above, which independently arrives at canonical action representations, execution-time authorisation boundaries, policy-and-state-bound decision records and replay-verifiable receipts.

What remains distinctive, and is offered for others to dispute:

1. **Provenance as a first-class, attested and fail-closed control variable.** Not "the record shows where the value came from", but: a resolver role that is not the requester signs a statement about origin; the verifier derives the class rather than adopting an assertion; an unattested claim degrades to P4; and the tier × class matrix is evaluated in the processing path.
2. **Binding to the rendered presentation**, not only to the canonical structure — with the rendering profile itself signed and versioned, and the rendered artefact retained.
3. **Cumulative delegation limits with reservation semantics**, rather than a per-action ceiling.
4. **Write-ahead evidence against the executing party**, rather than evidence written after the fact by the component with the strongest motive to omit it.

None of these four is claimed to be unprecedented. They are claimed to be the parts most often missing, and the profile exists so that the claim can be checked rather than asserted.

The composition is deliberate in a second sense: an organisation that already runs an identity provider, a policy engine, a service mesh and a SIEM has most of the parts. It is entirely legitimate to build the rest on top of what it has rather than buy a platform.

---

4. Threat model

Version 0.1 had a non-goals section and called it a threat model. A non-goals section is a liability disclaimer. This section names adversaries, states what is trusted, and gives every requirement in this document something to be traceable to.

4.1 Assets

Asset	Why an adversary wants it
The authority decision	Executing a consequential action without an authority decision, or with a widened one
The approval	Reusing, forging or redirecting a human authorisation
The parameters and their provenance	Making a falsified value look authoritative
The action tier	Escaping the controls indexed by tier
The evidence record	Suppressing, altering or reordering the history of what happened
Signing keys	Forging any of the above
Personal data in the evidence store	Surveillance, resale, coercion

4.2 Adversaries

#	Adversary	Capability assumed	Primary defences
A1	**Content author**	Controls a document, email, web page, tool description or record the agent will read. Cannot execute code in the control plane	§7.3 provenance attestation; tier ceilings; out-of-band verification; §18 presentation binding. **Not prevented** — see *Position Paper*, §3.2
A2	**Compromised business application**	Full control of a requesting application; can assemble arbitrary envelopes	§6 GI-2; §11 signer matrix; §7.2 tier derivation; §7.3 provenance attestation; §13 processing order
A3	**Compromised agent or model**	Produces arbitrary intents and parameters; may chain sub-agents	§6 GI-4; §17.2 delegation attenuation; §9 schema validation
A4	**Compromised enforcement point / broker**	Can execute, and can attempt to suppress or rewrite evidence	§11 separate signers; §20.2 write-ahead evidence; §20.3 sequence numbers with gap detection; §6 GI-8
A5	**Malicious or coerced approver**	Holds a valid approval credential	§18.3 quorum and role separation; §17.1 cumulative budget; detection, not prevention
A6	**Hostile tenant administrator**	Full control within one tenant	§11.4 key scoping by tenant; §6 GI-9; cross-tenant negative tests §24
A7	**Platform administrator**	Control of the operating domain	Key separation; independent witness (§20.4); four-eyes on key operations. **Bounded, not prevented**
A8	**Network adversary**	Intercepts, replays, reorders or delays messages	Signature over received octets (§12.6); §16 expiry and skew; §13 step 8 replay protection; channel binding (§22.4)
A9	**Target system**	May report a receipt that does not match what it committed	Out of scope. §26 records this as an open problem

4.3 Trust assumptions

The profile assumes, and cannot verify:

- the Workload Identity Authority correctly attests workloads;
- the Mandate Authority is not under the control of the requesting application;
- the Context and Provenance Resolver reads from the systems of record it claims to read from;
- the approver's credential is held by the approver;
- where a Credential Broker is used, it will not release a target credential without a valid execution proof, and it is not under the control of the enforcement point it constrains;

- where position (a) in §18.1.1 is taken, the Approval Service renders faithfully — the digest binds what that service rendered, not what the human saw;
- the Registry Authority's signing key is not under the control of a requesting application;
- key custody boundaries are what the deployment says they are;
- the target system's receipt describes what the target actually committed.

Each of these is a named assumption, not a proven property. A deployment that collapses two of these roles into one component removes the corresponding defence, and §5.2 states where that is permitted and where it is not.

4.4 Explicitly out of scope

Collusion between two or more separated authority roles. A fully compromised administrative domain with access to all key material and all evidence replicas. Truthfulness of source data. Correctness or fairness of a model output. Whether an approval was wise. Whether an organisation is compliant.

---

5. Roles

5.1 Role definitions

Role	Responsibility
Principal Resolver	Resolves the human or organisational principal and the legal or organisational context
Workload Identity Authority	Attests the acting agent, application or service instance
Mandate Authority	Issues, validates and revokes bounded assignments; maintains cumulative budget state
Context and Provenance Resolver	Supplies data classification, target context, parameter origins and assurance classes, **and attests them** (§7.3)
Registry Authority	Signs, versions and revokes the Action Type Registry: the mapping from action type, target and parameter thresholds to impact tier, the impact-determining parameter declaration, and the presentation profiles (§7.5)
Policy Decision Point (PDP)	Evaluates the action against versioned policy and returns a signed decision, including the derived tier
Approval Service	Under §18.1.1 position (a), renders the action per a declared presentation profile and presents it to an authorised human; under position (b), transports the canonical envelope and the signed presentation profile to a trusted approval client and never renders. In both positions it collects a signature from the approver's credential and never signs the approval itself
Policy Enforcement Point / Action Broker (PEP)	Revalidates and executes through controlled target credentials
Target Adapter	Maps the governed operation to target-native conditional, idempotent or compensating semantics
Evidence Service	Correlates, signs, checkpoints and serves decision and execution evidence
Credential Broker	Holds or mints the short-lived target credential and releases it only against a valid execution proof from the Evidence Service (§20.2.1). Where no such component exists — the case wherever the target does not expose a suitable verifier — the write-ahead control is detective rather than preventive
Witness / Timestamp Service	Provides independent checkpoint or time evidence

The Action Type Registry is new in 0.2 and exists solely so that the tier is not an assertion of the requester.

5.2 Separation requirements

Role separation is what makes the invariants true. Where roles collapse, defences are lost, and the profile is explicit about which collapses are acceptable.

- For actions at **tier 0 to 2**, one component MAY perform any combination of roles.
- For actions at **tier 3**, the Mandate Authority MUST NOT be under the operational control of the requesting application; the Context and Provenance Resolver MUST NOT be the requesting application or the agent (§7.3); and **the Evidence Service and its signing key MUST be outside the enforcement point's trust domain**, because §20.2.1 makes the execution proof a precondition of execution from tier 3 and a proof the enforcement point can issue to itself is not a control. Where a **preventive** write-ahead claim is made, the **Credential Broker** (or the verifying target) MUST likewise be outside that trust domain; where it is not, the claim is detective only.
- For actions at **tier 4 and 5**, the **Mandate Authority**, the **Action Broker**, the **Approval Service** and the **evidence signing key custody** MUST be separated into distinct trust domains with distinct key material. The Context and Provenance Resolver MUST NOT be the requesting application, the agent, or the Action Broker.
- For actions at **tier 5**, the **Witness / Timestamp Service** MUST be operated by a party other than the operator of the Evidence Service.

An implementation MUST publish its role-to-component mapping and the trust-domain boundary between them. A claim about tier 4 assurance from a deployment where broker and evidence service share a key is not a weaker claim; it is a different one, and it should be stated as such.

6. Control invariants and traceability

The control invariants are stated canonically in *Position Paper 01*, §4.4. Since v0.5.2 of that paper there are twelve rather than ten: **GI-5** was split into **GI-5A** and **GI-5H**, and **GI-6** into **GI-6P** and **GI-6D**, because this profile permits implementation profiles that deliver only the weaker half of each — see §18.1.1 and §15.2. An implementation states which of each pair it holds, per action type. **Tier 5 requires GI-5H and GI-6P.** This profile does not restate them in a second, differently-worded list — version 0.1 did that, producing ten invariants in the paper and fifteen in the annex, mapped onto nothing, so that an implementer could satisfy either and an assessor could not tell which applied.

Instead, every invariant traces to the rules, processing steps and negative tests that make it testable:

Invariant	Rules in this document	Processing step (§13)	Negative test (§24)
GI-1 Principal attribution	§9.2, §17.2	7	T31
GI-2 Authority separation	§5.2, §7.5, §11.1, §11.2, §17, §18.3	9, 11	T2, T18, T25, T28, T45, T46, T47
GI-3 Bypass declaration and detection	§15.4, §21.4	— (out-of-band reconciliation)	T10, T19
GI-4 Proposal is not authority	§9.3, §22.1	2, 6, 10, 12	T1, T17, T44
GI-5A Approval artefact equivalence	§9.4, §10.2, §10.3, §12.4, §18.2	13	T6, T20, T23, T27, T30, T32, T33, T34, T42, T44
GI-5H Human-view equivalence	§18.1.1 position (b)	13	T23 run against a locally-rendering client
GI-6P Preventive state freshness	§7.5, §9.6, §15.1, §15.2	14, 15	T4, T5, T29
GI-6D Detective state divergence	§15.1, §15.2, §15.4	— (reconciliation)	T19, T29, T49
GI-7 Revocation before commit	§13	9, 11, 15	T3
GI-8 Evidence independence	§20	16, 18	T9, T21, T35, T36, T48
GI-9 Tenant and key isolation	§11.4, §13.1	5, 8	T15, T26
GI-10 Safe failure	§21.3	— (declared per tier)	T13

A requirement in this document that traces to no invariant and to no threat in §4.2 is a candidate for deletion, and reviewers are invited to find them.

7. Impact tiers and provenance assurance

7.1 The two control variables

Every control in this profile is indexed by one of two variables: the **impact tier** of the action (0 to 5) and the **provenance assurance class** of each impact-determining parameter (P0 to P4). Both are defined in *Position Paper 01*, §2.4 and §2.5, together with the default tier × provenance matrix.

In version 0.1 both variables travelled in an envelope assembled by the requesting application. This meant the entire control model was indexed by two numbers chosen by the least trustworthy component in the system. A requester that declared tier 1 escaped the approval requirement, the segregation-of-duties rule, the assurance-mode requirement and the bypass rule while remaining formally conformant. A requester that declared `provenance_class: "P1"` for a value the model had hallucinated from an injected email defeated the matrix entirely. Neither required an attack on any cryptographic mechanism.

This is the single most important change in 0.2.

7.2 Tier derivation

1. The **Action Type Registry** MUST publish a signed mapping from `{action type, target system, parameter thresholds}` to an impact tier, with a version and an effective date.
2. The **PDP** MUST derive the tier from that registry at decision time and MUST include the derived tier, the registry digest and the registry version in the signed policy decision.
3. A tier value supplied by the requester in the envelope is **advisory only** and MUST be recorded as an assertion, not as context.
4. The **PEP** MUST recompute the tier from the same registry version before commit. A divergence between the requester's assertion, the PDP's derivation and the PEP's recomputation MUST fail closed and MUST be recorded as a distinct outcome, not as a generic denial.
5. Where a parameter threshold determines the tier — a payment amount, a number of affected records — the threshold evaluation MUST use the attested parameter value (§7.3), not the asserted one.

7.3 Provenance attestation

1. For every **impact-determining parameter** of an action at **tier 3 or above**, and — regardless of the tier finally derived — for every parameter whose value is capable of selecting tier 3 or above under any registry condition for that action type, the envelope MUST carry a detached attestation. The second clause closes a circularity: the tier is derived from a threshold on a parameter value, so a parameter that can move the action into the attested range must itself be attested before the derivation reads it (§13, step 10 precedes step 11) issued by a **Context and Provenance Resolver** that is not the requesting application or the agent.
2. The attestation MUST carry:

```
{
  transaction_id,
  parameter_name,
  resolutions: [                  # one per resolution path; P0 requires two
    { source_id,
      source_type,                # "system_of_record" | "asserted" | "derived"
      source_domain,              # organisational domain, for the independence test
      parameter_value_digest,     # {alg,value} over the canonical value commitment (§7.3.1)
      version_evidence,           # required for system_of_record
      asserter,                   # required for source_type = asserted:
      asserter_credential_ref,    #   subject, credential, method and evidence of
      auth_method,                #   the authentication that was recorded
      auth_evidence_digest,
      read_at }
  ],
  independence_basis,             # how source_domain values differ; see §7.4
  expires_at,
  resolved_class,                 # the resolver's own derivation, advisory only
  timestamp
}
```

`comparison` is **not** carried. Version 0.2.2 let the resolver assert that two resolutions agreed; a verifier that accepts that assertion is trusting the component the attestation exists to constrain. The verifier recomputes the comparison from the `parameter_value_digest` values.

P2 requires "an assertion by an authenticated person or service, with the authentication recorded". Version 0.2.2 had no field in which to record it, so P2 was derivable from an attestation that recorded nothing. The four asserter fields are mandatory for `source_type = asserted`.

and MUST be signed under a key bound to the resolver role (§11). Version 0.2.1 named singular fields — one source, one method — while requiring P0 to rest on two resolutions and their agreement. The array is what makes the derivation in §7.4 evaluable at all. 3. A verifier MUST NOT adopt an asserted `provenance_class`. It MUST evaluate the attestation and derive the class. **Where an attestation is required and is absent, unverifiable or expired, the parameter MUST be treated as P4**, which the default matrix denies at tier 4 and above. Below tier 3, where no attestation is required and none is present, the parameter carries `provenance_assessment: "not-assessed"` and **no** `derived_class` (§7.4); where an attestation is present the class is derived per §7.4. An implementation MUST NOT derive P4 for a tier 0–2 action solely because no attestation was required, and MUST NOT derive P2 from the absence of evidence. 4. The tier × provenance matrix MUST be evaluated as a processing step (§13, step 12). A prohibited combination MUST fail closed. Version 0.1 printed the matrix in the paper and referenced it from no invariant and no processing step, which meant a conforming implementation never had to apply it. 5. An implementation MUST publish which of its parameters are treated as impact-determining, per action type, in the same signed registry as the tier mapping (§7.5). This closes the gap that lets an implementation declare nothing material and remain conformant.

7.3.1 The parameter-value commitment

An attestation must be about **the value that is going to execute**, and until this revision nothing said so. The attestation carried a digest of a value; the envelope parameter carried a value; no processing step compared them. An envelope could therefore pair a valid P1 attestation for one amount with a different amount in the parameter, and the P1 class would be applied to a value nothing had attested. That defeats provenance as a control variable, which is the mechanism the whole profile rests on.

The commitment is canonical and typed:

```
parameter_value_digest =
  H( "GAIP-0.4/parameter-value/v1" || 0x00 ||
     canonical({ parameter_name, value_type, normalisation_profile, value }) )
```

`value_type` and `normalisation_profile` are required because equality of two decimal strings is not equality of two amounts: `1200.00`, `1200.0` and `1200` denote the same amount and hash differently. The normalisation profile fixes that — for a currency amount, the scale declared for the currency; for a string, a declared Unicode normalisation; for an identifier, a declared compaction. §12.3 defines the canonical decimal form.

A verifier MUST:

1. recompute `parameter_value_digest` from the envelope parameter's own value, type and normalisation profile;
2. reject the envelope unless **every** `resolutions[].parameter_value_digest` in the parameter's attestation equals the recomputed value — `E-PROVENANCE`;
3. perform this check at step 10 of §13, **before** tier derivation and **before** the tier × provenance matrix (step 12), because both consume the class;
4. apply the same comparison to `value_ref`, using the resolved plaintext commitment required by §9.4.

The recomputed comparison across resolutions is what yields `comparison` for §7.4: `equal` where every resolution's commitment matches the recomputed value, `differs` otherwise.

7.4 Derivation of the provenance class

An attestation carries evidence; the class is a function of that evidence, and the function MUST be deterministic. Two verifiers presented with the same attestation MUST derive the same class, or the matrix is not a control.

The verifier evaluates the attestation's `resolutions[]`, `comparison` and `independence_basis` against the **authoritative sources and freshness windows the Action Type Registry declares for that parameter** (§7.5). `resolved_class` from the resolver is advisory and MUST NOT be adopted.

Derived	Required conjunction
P0	Two or more resolutions; at least two against sources declared authoritative for that parameter; the two resolutions' `source_domain` values appearing as a declared independent pair in the entry's `independence_domains` for that parameter; the recomputed comparison `equal`; every reading inside its declared freshness window
P1	One resolution with `source_type = system_of_record`, against a source declared authoritative for that parameter, carrying `version_evidence` — an authenticated version, entity tag or state reference — and read inside its declared freshness window
P2	One resolution with `source_type = asserted`, by an authenticated person or service, with the authentication recorded
P3	One or more resolutions, all with `source_type = derived` by a model or parser from content not under the organisation's control — mail, documents, web content, tool descriptions, retrieved records
P4	Any of: an attestation is required and absent; signature or key resolution fails; the attestation is past its `expires_at`; the recomputed comparison is `differs`; a resolution's `parameter_value_digest` does not match the recomputed commitment (§7.3.1); a resolution names a source not declared authoritative where the claimed class requires one; a reading is outside its declared freshness window; or a field carries an unrecognised value

Rules of precedence. The **weakest** applicable class governs a parameter whose resolutions are in conflict; a confirmatory second resolution under P0 is not a conflict, which is the whole construction of P0. A system-of-record read **without** version evidence does not fall to P2 — an unversioned read is not an authenticated human assertion — it yields `not-assessed` if no attestation was required, and **P4** if one was. An unrecognised value in any field yields P4, never a default.

Assessment is a status, not a class. Version 0.2.1 derived P2 for a missing low-tier attestation, silently reclassifying a model-derived or untraceable value as an authenticated assertion. Version 0.2.2 fixed that by inventing a sixth class, PX — which left this profile with a taxonomy the position paper and the adoption guide do not share.

Both are wrong in the same way: "not assessed" is not a degree of provenance assurance. It is the absence of an assessment. So each parameter carries:

```
provenance_assessment: "assessed" | "not-assessed"
derived_class:         P0 | P1 | P2 | P3 | P4     # present only when assessed
```

- Where an attestation is **required** (tier 3 and above) the status is always `assessed`; an absent or invalid attestation derives **P4**, and the P4 column of the matrix governs.
- Where an attestation is **not required** (tiers 0 to 2) and none is present, the status is `not-assessed`, `derived_class` is **absent**, and the matrix is not consulted. The action proceeds only where the action type's policy explicitly permits an unassessed parameter at that tier, and that permission MUST be recorded in the evidence record.
- A parameter that is **not impact-determining** for its action type (§7.3, item 5) requires no attestation at any tier and carries `not-assessed` unless one is supplied. The attestation requirement is a property of the parameter's role in the action, not of the tier alone.
- A `not-assessed` parameter MUST NOT satisfy any requirement expressed as a class, and MUST NOT be presented in a rendering as though it had one.

The canonical taxonomy therefore remains **P0 to P4**, unchanged, in all three documents.

Independence, for P0

`independence_basis` is not free text. Each resolution carries a `source_domain`, and the registry declares which domains are mutually independent for a given parameter:

```
independence_domains: { parameter: [ [domain_a, domain_b], ... ] }
```

A resolution's domain is **not** taken from the attestation: the verifier resolves each `source_id` through the registry's signed `source_domains` mapping. Two resolutions are independent only where the resulting domains appear as a declared independent pair. Version 0.3 let the provenance resolver supply `source_domain` itself, which meant the component the attestation exists to constrain could label two reads of one system as two domains. A second read of the same system, or of two systems in the same declared domain, is not a second path — and version 0.2.2's "verifiable from the source identifiers" gave a verifier no way to establish that.

7.5 The Action Type Registry

Whoever controls the registry controls almost every tier-indexed rule in this profile. It is therefore an artefact with its own signer, its own lifecycle and its own change control, not a configuration file.

1. A registry snapshot MUST be a signed artefact carrying `{registry_id, version, effective_from, tenant_scope, entries[], previous_version_digest}`, signed by a **Registry Authority** whose key is scoped to that tenant and to this artefact type (§11.2). `version` is a monotonically increasing integer; `previous_version_digest` is the `GAIP-0.4/action-type-registry/v1` digest of the immediately preceding snapshot, and is `null` **only** in the genesis snapshot.
2. Each entry maps `{action_type, target, thresholds}` to a tier and declares, for that action type:

```
impact_determining: [ parameter names ],
authoritative_sources: { parameter: [ source ids ] },
provenance_freshness: { parameter: ISO-8601 duration },
required_preconditions: [
  { name, source, comparison, expected: { value_type, unit, scale, value | values },
    freshness_window }
]
# comparison: "equal" | "in_set" | "at_least" | "at_most"
# Comparison is by value_type, after decimal normalisation (§12.3) and a unit check.
# A type or unit mismatch fails closed with E-STALE; it is never coerced.
entry_id,                       # immutable, unique within the snapshot
condition,                      # §7.5.1
independence_domains: { parameter: [ [domain_a, domain_b], ... ] },
source_domains: { source_id: domain },   # signed mapping; a resolver cannot invent independence
presentation_profile_digest,    # {alg,value} over the signed profile object; §18.1
business_deduplication: {
  profile,                      # URI, referenced by execution_constraints
  key_fields: [ qualified names ],   # "parameters.<name>" or "intent.values.<name>"
  window                        # ISO-8601 duration
}
```

`presentation_profile_digest` replaces the mutable URI of 0.2.2: §18.1 requires the profile to be signed and versioned, and a URI that resolves to different content later is not a commitment. The complete signed profile object is published by the Registry Authority and addressed by this digest.

`authoritative_sources`, `independence_domains` and `provenance_freshness` are what make the derivation in §7.4 deterministic: without them, "a source declared authoritative for that field" has no referent and two verifiers can reach different classes from the same attestation.

3. Snapshots MUST be **revocable**, and a verifier MUST enforce a **minimum accepted registry version** from policy.
4. The PEP MUST NOT merely recompute the tier against the historical version named in the decision. At commit it MUST additionally verify that this version **remains accepted and has not been revoked**. A decision taken under a registry version withdrawn in the interval MUST fail closed.
5. A change that **weakens a control MUST require independent approval** — lowering a tier, removing an impact-determining parameter, weakening a presentation profile, removing or relaxing a required precondition, lengthening a freshness window, removing or broadening an authoritative source, altering an independence domain, weakening a deduplication key or window, or admitting a broader adapter assurance mode. Version 0.3 listed only the first three, which left the other six as unreviewed configuration changes with the same effect — a quorum under §18.3, by parties who are not the requesting application's operators — and MUST produce its own evidence record. Registry changes are governed actions in their own right.
6. `previous_version_digest` makes the registry a hash chain, so that a silently substituted history is detectable.

Without items 4 and 5 above, the tier derivation introduced in §7.2 relocates the problem rather than solving it: a requester that cannot assert its own tier but can influence the registry has the same power by a longer route.

7.5.1 Entry identity and the condition language

Version 0.2.2 mapped `{action_type, target, thresholds}` to a tier, which requires two entries for the same action type and target at different thresholds — and then §12.4 sorted entries by `action_type, target` and rejected duplicate sort keys, so the second entry was unrepresentable. The threshold vocabulary was also undefined: no operators, no overlap rule, no precedence, no default, no decimal normalisation.

a. Every entry MUST carry an immutable `entry_id`. Entries are sorted by `entry_id` (§12.4), so several entries may share an action type and target. b. An entry MUST carry a `condition`, expressed in this language and no other:

```
condition := { parameter, op, operand }
           | { all: [ condition, ... ] }
           | { any: [ condition, ... ] }
           | { always: true }

op       := "eq" | "ne" | "lt" | "lte" | "gt" | "gte" | "in"
operand  := { value_type, value }                       # scalar
          | { value_type, unit, scale, value }          # typed quantity
          | { value_type, values: [ ... ] }             # for "in"
```

Comparison is by `value_type`: exact decimal comparison for `decimal` after normalisation to the declared `scale` and check of the declared `unit`; code-point comparison for `string`; integer comparison for `integer`. A comparison between mismatched types or units MUST fail closed with `E-TIER`, never silently coerce.

c. **Matching is most-specific-first.** Where several entries match, the entry whose condition contains the greatest number of scalar comparisons **that must all hold** wins — that is, comparisons under `all` and at the top level count, and a disjunction under `any` counts as one, so a five-way `any` does not outrank a three-way `all`; where that ties, the snapshot is **ambiguous and MUST be rejected** with `E-TIER`. Version 0.3 resolved a tie by lowest `entry_id`, which made a security outcome depend on naming; a registry that cannot decide which tier applies should be fixed, not silently resolved. The algorithm is fully deterministic and requires no ordering convention outside the snapshot. d. Every `{action_type, target}` pair MUST have exactly one entry whose condition is `{always: true}`, serving as the **default**. A pair without one MUST fail closed with `E-TIER`. Version 0.2.2's `{"amount_gte":"1000.00"}` example defined no outcome when the condition was false. e. Overlapping conditions are permitted — the most-specific rule resolves them — but the Registry Authority SHOULD publish a coverage analysis with each snapshot.

`required_preconditions` closes a second gap. Version 0.2.1 required "at least one" precondition at tier 4 and 5, which an implementation could satisfy with an irrelevant version reference while omitting supplier status, available balance or sanctions state. The registry now declares the complete set that is material for the action type, and a verifier MUST check that the envelope carries all of them. Where an action type genuinely has no mutable material precondition, the entry MUST carry an explicit signed `no_material_preconditions` declaration with a rationale — never an artificial placeholder.

7.6 One term, not three

Version 0.1 used "material", "critical fields" and "impact-determining" for the same concept in three places. This edition uses **impact-determining** throughout, defined as: *a parameter whose value can change the impact tier, the policy outcome, the identity of the beneficiary, or the reversibility of the action.* Which parameters those are is declared per action type in the signed registry (§7.3, item 5) and is not a per-envelope decision by the requester.

---

8. Artefacts

A GAIP transaction consists of:

#	Artefact	Issued by	Purpose
1	**Intent Proposal**	Requesting application or agent	Structured proposed business action with origin marking
2	**Provenance Attestation** (one per impact-determining parameter, tier ≥ 3)	Context and Provenance Resolver	Establishes where a value came from (§7.3)
3	**Mandate Reference**	Mandate Authority	Bounded authority, with cumulative budget state
4	**Policy Decision Record**	PDP	Decision, derived tier, policy digest, inputs, obligations
5	**Approval Record**	The approver's credential, collected by the Approval Service	Signature bound to action, provenance, presentation, preconditions and expiry
6	**Governed Action Envelope**	PEP-facing composite	The short-lived object delivered to the enforcement point
7	**Commit Intent Record**	PEP, **before** execution	Write-ahead evidence (§20.2)
8	**Execution Receipt**	PEP, incorporating the target receipt	Outcome and committed target state
9	**Evidence Record**	Evidence Service	Correlated decision and execution proof with checkpoint status

An implementation MAY store these as separate signed objects referenced by digest, or as a signed composite. Where a composite is used, it **MUST embed each inner signed artefact byte-for-byte**, together with its own signature container, and MUST NOT reduce an inner artefact to a bare digest under a single outer signature.

This distinction matters and JWS terminology obscures it. **JWS General Serialization carries several signatures over one common payload**; it does not preserve independent signatures over separate inner objects. It is therefore the wrong container for a composite whose artefacts were signed by different roles at different times. The correct construction is a composite whose members are the **complete inner JWS or COSE objects as received**. In the JSON profile each member is carried as a **base64url-encoded octet string** with an accompanying `media_type` and `representation_family` (`json` or `cbor`); in the CBOR profile as a byte string with the same two fields. Version 0.2.2 required byte-for-byte preservation and defined no encoding in which to preserve them. The outer signature covers that structure — so that a verifier can extract each inner artefact and verify it against the key of its own issuing role, exactly as it would have done had the artefacts arrived separately. COSE_Sign with countersignatures per RFC 9338 is a suitable alternative where a genuine countersignature relationship exists. A verifier MUST be able to check each inner artefact against the key of its own issuing role.

The reason is stated plainly: if the broker signs a composite and the mandate, decision and approval survive only as digests, a broker compromise forges all three at once, and GI-2, GI-5A and GI-8 fall together with one key.

9. Envelope data model

9.1 Overview

Field	Cardinality	Requirement
`profile`	1	Exact profile revision, e.g. `"GAIP-0.4"`. MUST be listed in `critical` (§9.1.1)
`critical`	1	Array of field or extension names a verifier MUST understand or reject (§22.3)
`transaction_id`	1	Globally unique; UUIDv7 RECOMMENDED
`tenant_id`	1	Authority domain within which identities, policies, keys and evidence resolve
`issued_at`	1	Envelope issuance time (§16)
`not_before`	0..1	Earliest permitted commit time
`expires_at`	1	Envelope expiry (§16)
`principal`	1	Human or organisational principal; MUST NOT be only a shared technical account
`actor_chain`	1..n	Ordered delegation chain (§17.2)
`intent`	1	`{type, schema_uri, schema_digest, values, origin, principal_confirmation}` (§9.3). Bound through `intent_digest`
`intent_digest`	1	`{alg, value}` over the canonical intent (§10.2). Inside the approved-action digest
`mandate`	1	Reference, version, purpose, scope, budget state, delegation rule, validity, revocation reference
`context`	1	Legal entity, business scope, data class, **derived** action tier with registry reference, security posture
`asserted_tier`	0..1	Tier claimed by the requester. **Advisory only** (§7.2, item 3); recorded as an assertion and never used as context
`parameters`	0..n	Name, value or protected reference, `value_type`, `normalisation_profile`, `parameter_value_digest`, attestation reference and digest, `provenance_assessment`, derived class (§9.4)
`policy_decision`	1	Decision ID, effect, policy digest, policy input digest, **derived tier, action-type registry digest and registry version** (§7.2), decision time, obligations, expiry
`approvals`	0..n	Approval records (§18); cardinality set by policy quorum, not by the requester
`preconditions`	0..n	Impact-determining resource versions, states, balances, screening results. **Cardinality is set by the registry's `required_preconditions` for the action type** (§7.5): the envelope MUST carry every declared precondition, or the registry's signed `no_material_preconditions` declaration. An incomplete set fails closed at step 14 and makes GI-6P unenforceable
`freshness_window`	—	**Not an envelope-level field.** It is carried per precondition (§9.6) and is mandatory from tier 3
`execution_constraints`	1	Target, operation, `allowed_adapter`, derived idempotency key, `maximum_attempts`, `assurance_mode`, `evidence_profile_digest`, `business_deduplication_profile` where the action type declares one, and — **`where assurance_mode is broker-ledger`** — `reconciliation_interval` and `maximum_detection_latency` (§15.2). All of these except the derived idempotency key are inside the approved-action digest (§10.2), so a downgrade breaks the approval. `maximum_attempts` and `evidence_profile_digest` were outside it in 0.3, although raising the retry budget changes execution behaviour and changing the evidence profile changes what is provable
`evidence_profile`	1	Canonicalisation, signature, timestamp, checkpoint and retention profile
`break_glass`	0..1	Present only for a declared emergency path (§19)

Version 0.1 omitted `issued_at`, `not_before`, `critical`, `freshness_window` and `presentation_digest`, while requiring elsewhere that a verifier enforce a maximum clock skew, reject unknown critical features, honour a "permitted freshness window" and validate a presentation digest. Four of its mandatory processing steps referred to fields that did not exist.

9.1.1 The profile identifier carries the requirements revision

`profile` MUST carry the **exact** revision — `"GAIP-0.4"` for this document — and MUST be listed in `critical`. It is inside the approved-action digest input scope (§10.2), so it is covered by the approval signature and cannot be stripped or rewritten without breaking the binding. A verifier MUST enforce a **minimum acceptable profile revision** from policy (§22.3) and MUST reject anything below it.

Version 0.2.1 kept `"GAIP-0.2"` on the wire for both revisions while adding mandatory fields, changed derivation semantics and new registry behaviour. The consequence was that a policy could not require the revised semantics, downgrade protection could not distinguish the two, a capability statement could claim 0.2.1 while every envelope it received identified only 0.2, and an old and a revised verifier could apply different rules to the same declared profile. A security-relevant revision that is invisible on the wire is not a revision a verifier can rely on.

9.2 Principal and actor

`principal` MUST identify a natural person or an organisational body, with the legal entity. A shared technical account is not a principal.

`actor_chain` is an ordered array from the outermost actor to the one that will execute. Each element carries `{identity, identity_type, instance_id, attested_by, attestation_ref}`. Single-element chains are permitted; the field is an array so that the common topology — planner calls executor calls sub-agent calls tool server — is expressible at all. Version 0.1's single `actor` object could not represent it, which made GI-1 unsatisfiable for any real agent deployment.

9.3 Intent

`intent.origin` MUST be one of `model-generated`, `user-confirmed`, `system-generated`. For tier 3 and above, `model-generated` alone MUST NOT proceed to a policy decision: schema validation and principal confirmation are required first, and the confirmation is recorded.

`intent` MUST carry `{type, schema_uri, schema_digest, values, origin, principal_confirmation}` and MUST be bound into the approved action through an `intent_digest` under `GAIP-0.4/intent/v1` (§10.2).

`schema_digest` is required because a URI is not a commitment: content behind a stable URI can change after approval. Values outside the schema MUST be rejected, not interpreted.

Version 0.2.2 bound only `intent{type, schema}` into the approved-action digest. Two materially different intents sharing a type and a schema URI therefore produced the same digest whenever their parameter arrays happened to match, and the intent's origin marking and the principal's confirmation record were outside the approval entirely — so `model-generated` could be rewritten to `user-confirmed`, or a confirmation substituted, without breaking any binding. That is a defect in GI-4 as much as in GI-5A.

9.4 Parameters

Each parameter carries `{name, value | value_ref, value_type, normalisation_profile, parameter_value_digest, attestation_ref, attestation_digest, provenance_assessment, derived_class}`. `value_type` and `normalisation_profile` are what make §7.3.1 recomputable at all; `provenance_assessment` is `"assessed"` or `"not-assessed"` and `derived_class` is present only in the first case (§7.4). `attestation_digest` is the digest of the signed attestation artefact under `GAIP-0.4/provenance-attestation/v1`, and it — not the mutable `attestation_ref` — is what the approved-action digest binds. Without it, a different attestation asserting the same value and class could be substituted after approval without breaking any digest. Note what is absent: there is no requester-supplied `provenance_class` and no requester-supplied `verified_against`. Both were self-asserted fields in 0.1 and are replaced by `attestation_ref`, which points at a signed statement from a resolver role, and `derived_class`, which the verifier computes and the requester cannot set.

Impact-determining parameters MUST be present in clear text to the PDP and to the approver. A bare digest defeats both: the PDP cannot evaluate `amount ≤ remaining_budget` against a hash, and an approver who is shown a hash has approved nothing. Where confidentiality requires it, `value_ref` MAY carry an encrypted disclosure resolvable by the PDP and the Approval Service, or a **salted commitment** in the manner of SD-JWT or BBS+.

A `value_ref` MUST be **immutable and content-addressed**, and MUST be accompanied by a commitment to the resolved plaintext — the salted commitment itself, or a digest under `GAIP-0.4/parameter-value/v1`. A mutable locator is not acceptable: a reference such as `store://tenant-42/param/9931` that resolves to one value at approval time and another at commit time binds the approval to a pointer rather than to a value, and defeats GI-5A without triggering any check. A verifier MUST reject a `value_ref` whose resolved plaintext does not match the accompanying commitment. An **unsalted digest MUST NOT** be used for a value drawn from a small or guessable domain — an IBAN, a payment amount, a salary, a date of birth — because it is reversible by enumeration in seconds and is not a privacy measure.

9.5 Mandate

Beyond identifier, version, purpose, scope, validity and revocation reference, the mandate carries `budget` (§17.1) and `delegation` (§17.2).

9.6 Preconditions and freshness

Each precondition carries `{name, resource, addressing_scheme, expected_version | expected_state, read_at, freshness_window}`. `name` MUST match the `name` of the corresponding entry in the registry's `required_preconditions` (§7.5); `source` in that entry resolves to `{resource, addressing_scheme}` here. Without a common key, a verifier cannot perform step 14 at all. `addressing_scheme` MUST name how the resource identifier is to be interpreted by the target adapter, because two implementations addressing "the supplier record" in different schemes cannot compare preconditions at all.

`freshness_window` bounds how old a precondition reading may be at commit. It is **per precondition**, and it is **mandatory from tier 3 upward**; where the envelope omits it, the verifier MUST take the value declared for that precondition in the registry entry (§7.5), and MUST reject the envelope where neither carries one.

Version 0.2.1 referred to a "profile default" that this document never defined. There is no profile default. An unbounded freshness window makes GI-6P satisfiable by a reading of any age, which is the same as not checking.

The verifier MUST also check that the envelope carries **every** precondition the registry declares as required for the action type, or the registry's signed `no_material_preconditions` declaration.

9.7 Approval record

Each element of `approvals` carries:

Field	Requirement
`approver`	Identity of the approving natural person
`approver_role`	The role under which the approval is given, for quorum and segregation evaluation (§18.3)
`auth_method`, `auth_assurance_level`	How the approver authenticated
`auth_time`	Time of authentication
`session_binding`	Device or session binding where available
`approver_credential_ref`	The credential under which the approval was signed, resolvable to a key, a role and a validity period
`approved_action_digest`	§10.2
`approved_precondition_digest`	§10.2
`presentation_digest`	§10.2, §18.1
`policy_decision_digest`	§10.2
`nonce`	Single-use value binding this approval to this transaction
`expires_at`	Approval validity (§16)
`signature`	Under the approver's own credential (§11.1), over the binding scope in §10.3

`presentation_digest` was absent from the version 0.1 data model while its processing order required a verifier to check it. It is a first-class field here.

10. Digest definitions and domain separation

Version 0.1 named four digests — `approved_action_digest`, `approved_precondition_digest`, `policy_digest`, `value_digest` — and defined none of them: no input scope, no canonicalisation subset, no algorithm identifier, no domain separation. Two vendors hashing different projections of the same envelope are both conformant and neither can verify the other. Worse, without domain separation a precondition digest is a syntactically valid action digest.

10.1 Construction

Every digest in this profile is constructed as:

```
digest = H( label || 0x00 || canonical_bytes )
```

where `label` is the exact UTF-8 label from §10.2, `0x00` is a single zero octet, and `canonical_bytes` is the canonical serialisation (§12) of the substructure defined for that label.

A digest MUST be carried as the object

```
{"alg": "sha-256", "value": "<lowercase hex>"}
```

This is the **only** permitted representation. Multihash, a bare hex string, and a prefixed string such as `sha256:…` MUST NOT be used. Version 0.2.2 permitted two representations and then used a third in its own test vectors; one representation, used everywhere including the vectors, is what makes an implementation reproducible.

Where a digest is an input to another digest, the `{alg, value}` object is canonicalised as an ordinary object member — there is no separate binary encoding.

`H` for this edition is SHA-256. SHA-384 and SHA-512 are permitted where declared in the evidence profile.

10.2 Labels and input scopes

Label	Input scope
`GAIP-0.4/approved-action/v1`	`{profile, transaction_id, tenant_id, intent_digest, parameters[…], execution_constraints{…}}` where each parameter is `{name, value or value_ref, value_type, unit?, normalisation_profile, parameter_value_digest, attestation_digest?, provenance_assessment, derived_class?}` and the execution constraints are `{target, operation, allowed_adapter, assurance_mode, business_deduplication_profile, evidence_profile_digest, maximum_attempts, reconciliation_interval?, maximum_detection_latency?}`. **Presence rules:** `unit` is present for a typed quantity and absent otherwise; `attestation_digest` and `derived_class` are present exactly when `provenance_assessment` is `"assessed"` and absent when it is `"not-assessed"`; the two reconciliation fields are present exactly when `assurance_mode` is `broker-ledger`. A field that is absent under these rules is omitted from the canonical object, not carried as null
`GAIP-0.4/intent-schema/v1`	`{dialect, media_type, schema}` — the canonical schema document with its media type and dialect, excluding any detached signature container. This is the input to `intent.schema_digest`
`GAIP-0.4/presentation-profile/v1`	The complete canonical presentation-profile payload, excluding signature container. Input to `presentation_profile_digest` (§7.5, §18.1)
`GAIP-0.4/authentication-evidence/v1`	`{subject, credential_ref, method, issuer, time, assurance_level, session_binding}`. Input to `auth_evidence_digest` (§7.3)
`GAIP-0.4/credential-scope/v1`	`{tenant_id, transaction_id, target, operation, resource_constraints}` — the credential scope **derived by the issuer** from the approved action, not supplied by the requester. Input to `credential_scope_digest` (§20.2.3)
`GAIP-0.4/approved-preconditions/v1`	`{transaction_id, tenant_id, preconditions[]}`, each element carrying its own `freshness_window`
`GAIP-0.4/presentation/v1`	The rendered presentation artefact (§18.1)
`GAIP-0.4/policy/v1`	The policy source bundle as evaluated
`GAIP-0.4/policy-decision/v1`	`{decision_id, effect, policy_digest, policy_inputs_digest, derived_tier, registry_digest, registry_version, decision_time, obligations, expires_at}`, excluding the signature container. This is the input to `policy_decision_digest` in §9.7 and §10.3

Label	Input scope
`GAIP-0.4/policy-inputs/v1`	The attribute set supplied to the PDP
`GAIP-0.4/envelope/v1`	The whole envelope excluding signature containers **and excluding** `execution_constraints.idempotency_key`, which is derived from this digest (§13.1) and would otherwise be an input to itself
`GAIP-0.4/parameter-value/v1`	`{parameter_name, value_type, normalisation_profile, value}` (§7.3.1). For a salted commitment the object additionally carries `salt`
`GAIP-0.4/intent/v1`	`{type, schema_uri, schema_digest, values, origin, principal_confirmation}` (§9.3)
`GAIP-0.4/execution-proof/v1`	The complete canonical ExecutionProof, excluding its signature container (§20.2.3)
`GAIP-0.4/idempotency-key/v1`	`{tenant_id, target, operation, envelope_digest}` (§13.1)
`GAIP-0.4/action-type-registry/v1`	The complete canonical registry snapshot, excluding signature containers. This is the input to `registry_digest`, and `previous_version_digest` is this digest of the immediately preceding snapshot
`GAIP-0.4/provenance-attestation/v1`	The complete canonical attestation (§7.3, item 2), excluding signature containers. This is the input to `attestation_digest`
`GAIP-0.4/mandate/v1`	The complete canonical mandate artefact, excluding signature containers
`GAIP-0.4/commit-intent/v1`	The complete canonical Commit Intent Record, excluding signature containers
`GAIP-0.4/execution-receipt/v1`	The complete canonical execution receipt, excluding signature containers
`GAIP-0.4/evidence-record/v1`	The complete canonical evidence record, excluding signature containers

Four further labels are new in 0.4: the intent schema, the presentation profile, the authentication evidence and the credential scope. Each was a mandatory digest field in 0.3 with no defined construction, so two implementations could hash different objects while both following the prose — and each of the four determines something load-bearing: schema immutability, what the approver is shown, whether P2 is derivable, and what credential an execution proof releases.

Six labels are new in 0.2.2. Version 0.2.1 made `registry_digest`, `previous_version_digest` and `attestation_digest` mandatory fields while defining no construction for any of them — so two implementations could legitimately hash the unsigned payload, the whole JWS object, or the snapshot with or without its signature, and the tier derivation would depend on a digest the profile never defined.

`policy_digest` alone is insufficient and 0.1 required only that. A policy digest identifies the rules; it says nothing about the attributes they were evaluated against. Both MUST be present, and `policy-inputs` is what makes a decision reproducible.

10.3 Approval binding scope

The approval signature (§18.2) covers a distinct structure, not merely the action digest:

```
{profile, transaction_id, tenant_id, mandate{id, version}, policy_decision_digest,
 approved_action_digest, approved_precondition_digest, presentation_digest,
 approver, approver_role, approver_credential_ref,
 auth_method, auth_assurance_level, auth_time, session_binding,
 expires_at, nonce}
```

`approver_role`, `auth_assurance_level`, `session_binding` and `approver_credential_ref` were **outside** the signature in 0.2.1 while the quorum rule, the segregation rule and any authentication-strength condition were evaluated against them. A record assembler could therefore change the role under which the person allegedly approved, downgrade the recorded assurance level, drop the device binding, or point the record at a different credential — none of which invalidated the approver's signature. They are inside it now.

The inclusion of `transaction_id`, `policy_decision_digest` and `nonce` closes a defect in 0.1: two identical recurring payments produce identical action digests, so an approval that covered only "the action" could be captured and applied to a *different* transaction with the same parameters. Version 0.1's negative tests covered replay with modified parameters and exact envelope replay, but not this case. It is now test T20.

Approvals MUST be single-use. The Approval Service MUST enforce this and MUST record the attempt when a used approval is presented again.

11. Signer matrix

11.1 Who signs what

Artefact	Signing role	Requirement
Action Type Registry snapshot	**Registry Authority**	MUST
Provenance Attestation	Context and Provenance Resolver	MUST at tier ≥ 3
Mandate Reference	Mandate Authority	MUST
Policy Decision Record	PDP	MUST
Approval Record	The approver's own credential	MUST at tier ≥ 4; SHOULD at tier 3
Governed Action Envelope	Assembling component (usually the PDP or a broker-facing composer)	MUST
Commit Intent Record	PEP	MUST at tier ≥ 3
ExecutionProof	Evidence Service	MUST at tier ≥ 3 (§20.2.3)
Execution Receipt	PEP	MUST
Evidence Record and checkpoints	Evidence Service, under key custody separated from the PEP at tier ≥ 4	MUST

In 0.1 the `approver` field was an identifier, not a key. Nothing required the approval to be a signature under the approver's own credential, so an assertion by the Approval Service satisfied the text — which reduced approval forgery to the compromise of one service. The Approval Service is now a collector and a renderer, not a signer of approvals.

11.2 Verification of signer authority

A verifier MUST check, for every signature, that the key identifier resolves to `{role, tenant, validity period, permitted artefact types}` and that the signer is authorised **for this artefact type and this tenant**. A valid signature from a correctly-provisioned key in the wrong role or the wrong tenant MUST be rejected.

11.3 Algorithm rules

- An implementation MUST maintain an explicit algorithm allowlist.
- `alg: none` MUST be rejected.
- The algorithm MUST be selected from policy and validated against the expected key type. It MUST NOT be taken from the message header alone.
- Symmetric and asymmetric confusion MUST be excluded: a key registered for an asymmetric algorithm MUST NOT be usable to verify a MAC, and the reverse.

11.4 Key scoping

Key material MUST be scoped to a tenant and a role. A cross-tenant key reference MUST be rejected at step 5 of §13 and MUST be recorded as a security event, not a validation error.

12. Serialisation, canonicalisation and verification order

12.1 JSON profile

- I-JSON compatible data model (RFC 7493).
- RFC 8785 JSON Canonicalization Scheme for the canonical form.
- JWS (RFC 7515) as the signature container, **one per artefact**. JWS General Serialization carries several signatures over one common payload and MUST NOT be used to carry a composite of separately-signed artefacts; see §8 for the correct composite construction.
- Duplicate object member names MUST be rejected **in the lexer**, before the object is materialised. A last-wins parser discards the duplicate before any check can see it.

12.2 CBOR profile

- Deterministic CBOR encoding.
- COSE_Sign1 or COSE_Sign per RFC 9052; countersignatures per RFC 9338 for multi-party approval.
- Optional RFC 3161 timestamp integration through an explicitly declared wrapper.

12.3 Numbers, amounts and identifiers

Monetary amounts, quantities and identifiers MUST be carried as strings, with a declared decimal grammar, and MUST NOT be carried as JSON numbers.

RFC 8785 serialises numbers as ECMAScript doubles. The consequences fall on precisely the fields this profile exists to protect: decimal amounts do not survive the round trip without loss, and identifiers above 2^53 are silently corrupted. Version 0.1 required implementations to "define number representation" and then did not do so, while carrying `{"amount": 5000}` in its own example envelope.

The decimal grammar for this edition: an optional leading `-`, one or more digits, an optional `.` followed by one or more digits. No exponent, no thousands separator, no leading `+`. A currency amount MUST carry its **unit** as a field of the parameter itself — `unit: "EUR"` — not only as a sibling `currency` parameter. Version 0.3 put a unit on the registry's operand and none on the parameter it compared against, so a numerically identical amount in another currency had no defined comparison behaviour at all.

Where an action type expresses the currency as a separate parameter, the registry condition MUST name it through `unit_parameter`, and the verifier MUST check that the parameter's own `unit` and the named currency parameter agree. A disagreement fails closed with `E-TIER`.

For the cumulative budget, the registry entry MUST name `budget_amount_parameter` and `budget_currency_parameter`. A reservation MUST fail closed with `E-BUDGET` where the action currency differs from the mandate currency. **Cross-currency reservation is not permitted by default.** Where a deployment allows it, the policy MUST require an FX conversion record carrying the rate, its authoritative source, the reading time, the direction and the rounding rule, and that record MUST be part of the evidence. Comparison is exact decimal comparison, not floating-point comparison.

Canonical decimal form. The grammar above admits `1`, `1.0`, `1.00` and `01.00` as distinct strings for the same quantity, and `-0` as a distinct signed zero. Before any tier evaluation, budget arithmetic, deduplication key construction or parameter-value commitment (§7.3.1), a decimal MUST be normalised to its canonical form: no leading zeros other than a single `0` before the point; the exact number of fractional digits given by the declared **scale** for its type or currency, zero-padded or rejected if it would require truncation; and no `-0`. Two decimals are equal only if their canonical forms are byte-identical. A value that cannot be normalised without loss MUST be rejected, never rounded.

The string rule applies to exact decimals and to identifiers that may exceed 2^53. It does **not** apply to bounded structural integers defined by this profile — `max_depth`, `velocity.max_actions`, `mandate.version`, `tier`, registry `version`, `operand.scale`, evidence sequence numbers, retry budgets and size limits. Those are JSON numbers, MUST be integers in the range 0 to 2^53−1, and MUST NOT carry a fractional part or an exponent. A verifier MUST reject a structural integer outside that range rather than accept a lossy value.

12.4 Array ordering

JCS and deterministic CBOR both preserve array order; neither imposes one. An ordering rule is therefore required, or two implementations that agree on every value will produce different digests.

Array	Rule
`actor_chain`	**Semantic.** Order is the delegation order, outermost first, and MUST be preserved
`parameters`	**Not semantic.** MUST be sorted by `name`, ascending, by UTF-8 code point, before digest construction
`preconditions`	**Not semantic.** MUST be sorted by `name`, ascending
`approvals`	**Not semantic.** MUST be sorted by `approver`, then `auth_time`, ascending
`critical`	**Not semantic.** MUST be sorted ascending
`obligations` (§10.2 policy-decision scope)	**Not semantic.** MUST be sorted by `type`, then by canonical serialisation, ascending. Without this, two policy decision points emitting the same obligations in a different order produce different `policy_decision_digest` values, and that digest is inside the approval binding
`delegation.attenuations`	**Semantic.** Order is the attenuation order and MUST be preserved
`resolutions` (§7.3)	**Not semantic.** MUST be sorted by `source_id`, then `read_at`, ascending
`required_preconditions` (§7.5)	**Not semantic.** MUST be sorted by `name`, ascending
`entries` (§7.5)	**Not semantic.** MUST be sorted by `entry_id`, ascending. Sorting by `action_type` and `target`, as 0.2.2 required, made a second threshold entry for the same pair unrepresentable
`condition.all`, `condition.any` (§7.5.1)	**Not semantic.** MUST be sorted by canonical serialisation of each member, ascending
`operand.values` (§7.5.1)	**Not semantic.** MUST be sorted ascending
`independence_domains` pairs (§7.4)	**Not semantic.** Each pair MUST be sorted internally, and the list of pairs sorted ascending
`authoritative_sources` values, `impact_determining`, `key_fields`	**Not semantic.** MUST be sorted ascending

Where an array is not semantic, a duplicate sort key MUST be rejected rather than resolved by insertion order.

12.5 Cross-representation digests

A digest computed over the JSON canonical form and a digest computed over the CBOR canonical form of the same logical object **are different values**. This profile does **not** define a mapping between them. Cross-representation digest comparison is therefore **prohibited**: an implementation MUST record which representation family a digest was computed over, and a verifier MUST reject a comparison across families rather than attempt a conversion. A future edition may define the mapping; until it does, a transaction stays within one representation family end to end.

12.6 Verification before canonicalisation

The signature MUST be verified over the octets as received. An implementation MUST NOT canonicalise input and then verify a signature over the canonicalised result.

Version 0.1 canonicalised at processing step 2 and verified signatures at step 3. That ordering lets a recipient *repair* a malformed or ambiguous message into a valid one — the classic canonicalisation gadget. Combined with the prohibition on duplicate field names, it produced a defence that could never fire: the parser discarded the duplicate before anything checked for it.

The required order is: **parse with hardening → reject non-canonical, duplicate or unknown-critical input → verify signatures over the received octets.** Non-canonical input is rejected, not normalised. Where an implementation must re-serialise for storage, it MUST retain the original octets for as long as the signature must remain verifiable.

---

13. Processing procedure

A verifier or enforcement point processes an envelope in this order. Steps 1 to 15 are validation; a failure at any of them MUST NOT fall through to execution.

#	Step	Notes
1	**Parse with hardening.** Enforce size, depth, array-length and member-count limits. Reject duplicate member names in the lexer	§22.5
2	**Validate local framing only** — the structure needed to locate signature containers, `critical` and `profile`. **No external resolution of any kind.** Full profile-schema validation happens here where the schema is local; nothing that requires fetching does	§22.1
3	**Reject non-canonical input, unknown critical features, and any profile revision below the minimum accepted**	§12.6, §22.3, §9.1.1
4	**Verify signatures over the received octets**, resolve key identifiers to role, tenant and validity, and verify signer authority per artefact type	§11
5	**Validate envelope time.** Reject where `issued_at` lies outside the permitted skew, where `not_before` has not been reached, where `expires_at` has passed, or where `expires_at` precedes `issued_at`	§16
6	**Resolve the content-addressed intent schema, verify `intent.schema_digest` against it, recompute `intent_digest`, and perform semantic validation.** This is the first step permitted to resolve anything the requester named, because it is the first step after the envelope is authenticated. At tier 3 and above, reject a `model-generated` intent carrying no recorded principal confirmation	§9.3, §10.2
7	**Resolve principal and actor chain status**; verify attestations for every hop	§9.2, §17.2
8	**Verify transaction uniqueness and idempotency state, and reserve the business deduplication key** — after authentication, never before	§13.1, §13.1.1
9	**Validate the mandate chain** — purpose, scope, validity, revocation, delegation attenuation — and **reserve** against the cumulative budget	§17
10	**Recompute every `parameter_value_digest`** from the envelope parameter's own value, `value_type` and `normalisation_profile`, and compare it against each resolution's digest in that parameter's attestation. A mismatch fails closed with `E-PROVENANCE`. This precedes tier derivation and the matrix because both consume the class	§7.3.1
11	**Validate the policy decision** — signature, policy digest, policy-input digest, obligations, expiry, and the derived tier against a recomputation from the registry, whose version MUST still be accepted and unrevoked. **The effect MUST be `permit`.** An effect of `deny`, `defer` or `indeterminate`, an unrecognised effect, or no effect at all MUST fail closed with `E-POLICY`. Every blocking or pre-execution obligation MUST be discharged and evidenced before step 16	§7.2, §7.5, §13.4
12	**Evaluate the tier × provenance matrix** using classes derived from the attestations per §7.4. A prohibited combination fails closed	§7.3, §7.4
13	**Validate approvals** — quorum, role separation, approver authority, single use, and the full binding scope including `presentation_digest`	§18, §9.7, §10.3
14	**Resolve current impact-determining state** and compare with **every** registry-declared precondition, each within its freshness window	§7.5, §9.6
15	**Re-evaluate policy** where a declared input changed or where policy requires commit-time evaluation. If the re-evaluation differs, the approval is invalidated and MUST NOT stand	§13.2
16	**Write the Commit Intent Record, checkpoint it, and obtain the one-time execution proof** from the Evidence Service before any side effect	§20.2
17	**Execute once** through the declared adapter and assurance mode, presenting the execution proof; obtain the target receipt	§15, §20.2
18	**Emit and checkpoint the Execution Receipt and Evidence Record**; commit, release or quarantine the budget and deduplication reservations	§20, §17.1, §13.1.1
R	**Rollback.** On any failure from step 9 through step 16 — from the moment the first reservation exists until a target side effect becomes possible — the verifier MUST release whichever reservations exist, and MUST record the release	§13.3

Eight changes from 0.1's thirteen-step order are structural rather than cosmetic. **Nothing the requester named is resolved before the envelope is authenticated** — step 2 is local framing only and schema resolution waits for step 6. The parameter-value commitment is recomputed at step 10, before anything consumes a provenance class. And the policy effect is checked at step 11, which 0.3 omitted entirely. Schema and intent-schema validation is an explicit step (step 2). Canonicalisation is gone as a separate step and replaced by rejection (step 3). **Envelope time validation is an explicit step (step 5)** — 0.2.1 carried `issued_at`, `not_before` and `expires_at` and required a verifier to check none of them. Idempotency checking moved after authentication (step 7). Write-ahead evidence was inserted before execution (step 16), because evidence created only after execution can be suppressed by the component that executed.

13.3 Rollback of reservations

Reservations are acquired early — the deduplication key at step 8, the budget at step 9 — because both are authority state that later steps depend on. Step 9 can itself fail, and everything from step 10 to step 16 can fail after it.

Version 0.2.2 sent those failures away from execution and defined no cleanup. §17.1 said that a denied, stale or revoked outcome releases the budget, while the processing table contained no operation that performed the release, and the deduplication reservation had no release path at all. The consequences are concrete: leaked budget that never returns, a business deduplication key permanently blocking a legitimate retry, and a denial-of-service path in which invalid envelopes consume a mandate's capacity.

1. On any failure from step 9 through step 16 — that is, from the moment the first reservation exists until a target side effect becomes possible — the verifier MUST release **whichever reservations exist** and MUST record the release as part of the terminal evidence record. A mandate or budget rejection at step 9 releases the deduplication key acquired at step 8; omitting that case, as 0.2.2 did, permanently blocks a legitimate retry of exactly the transaction the mandate check rejected.
2. Only an outcome for which target commitment is **genuinely unknown** — `Indeterminate`, `AwaitingReceipt` past its window, `Partial`, `Escalated` — may quarantine rather than release (§17.1).
3. A verifier MUST NOT quarantine on a validation failure. A failure at step 11 means the transaction never reached the target, and treating it as unknown consumes authority that was never exercised.
4. Reservation release MUST be idempotent, so that a broker restart between the two reservations, or between reservation and release, converges. An orphaned reservation whose transaction has no terminal record MUST be surfaced by the completeness monitor (§20.2.2), not silently expired.

13.4 Policy effect and obligations

The Policy Decision Record carries an `effect`. Version 0.3 bound it into the digest, distinguished `Evaluated → Denied` from `Evaluated → Authorised` in the state machine, and then never required any processing step to read it. A correctly signed, unexpired decision whose effect was `deny` satisfied every check in steps 11 to 17 and could reach execution — most easily below tier 4, where no approval record is required to raise a second failure.

1. `effect` MUST be one of `permit`, `deny`, `defer`, `indeterminate`. A verifier MUST reject an unrecognised value rather than treat it as permissive.
2. **Only `permit` may proceed.** Anything else moves the transaction to `Denied`, releases both reservations (§13.3) and MUST NOT obtain an execution proof.
3. `obligations` are typed and each carries `blocking: true | false`. A **blocking** obligation MUST be discharged, and its discharge evidenced, before step 16. An undischarged blocking obligation fails closed with `E-POLICY`.
4. A non-blocking obligation MUST still be recorded in the evidence record; silently dropping it is a defect, not a degradation.

13.1 Idempotency keys

The idempotency key **MUST be derived**, not supplied:

```
idempotency_key = H( "GAIP-0.4/idempotency-key/v1" || 0x00 ||
                     canonical({ tenant_id, target, operation, envelope_digest }) )
```

Version 0.2.2 concatenated the fields with zero-octet separators and no domain label. Raw concatenation of variable-length fields is a collision surface whenever a field may itself contain the separator, and an unlabelled digest can be confused with any other unlabelled digest in the profile. Hashing a canonical structured object under an explicit label removes both problems, and adding `operation` scopes the key to the operation as well as the target.

In 0.1 the key was an attacker-choosable field checked at step 4, before the principal was resolved at step 5. That made it an oracle: an unauthenticated party could probe for the existence and outcome of another party's transactions, and a colliding key could return another transaction's result. Deriving the key removes the choice; namespacing it by tenant and target removes the collision; checking it after authentication removes the oracle.

Lookup MUST be scoped to the tenant. A repeated request with the same derived key MUST return the existing outcome or fail closed, and MUST NOT create a second side effect.

13.1.1 Transport replay is not business duplication

The derived key prevents the **same envelope** from committing twice. It does not prevent the **same business operation** from being issued twice under two different `transaction_id` values — which is what actually produces a double payment, a duplicated purchase order or a second account.

An action type whose target operation is not naturally idempotent MUST therefore additionally declare a **business deduplication key** in the Action Type Registry: a deterministic function of the semantically identifying parameters of that operation — for a supplier payment, typically `{tenant, payee account, invoice reference, amount, currency, period}` — scoped to a declared window.

The broker MUST **reserve** the business deduplication key at step 8 alongside the transport-key check. A check is not sufficient: two concurrent envelopes can both look up the key, both observe no existing operation, and both execute. The reservation MUST therefore be atomic, and scoped to `{tenant, target, operation, deduplication window}`.

The reservation follows the same lifecycle as the budget reservation in §17.1 — `reserved → committed`, `reserved → released`, `reserved → quarantined` — and for the same reason: an indeterminate outcome MUST NOT release the key on a timer, because the target may still commit afterwards.

A collision on an active reservation MUST fail closed and be recorded as `E-DUPLICATE`, distinct from `E-REPLAY`. The two keys answer different questions and neither substitutes for the other.

13.2 Re-evaluation invalidates approval

Where step 15 produces a decision that differs from the decision the approval was bound to, the approval is void. The transaction MUST take the `Revalidation → Denied` transition and MUST NOT execute on the strength of the earlier approval. (`Stale` is reserved for expiry and precondition mismatch; a policy re-evaluation that denies is a denial, not staleness.) Version 0.1 required re-evaluation at its step 10 and then proceeded to execution at step 11 without rechecking the approval against the new decision — so a policy change that would have denied the action left the approval standing.

---

14. Processing state machine

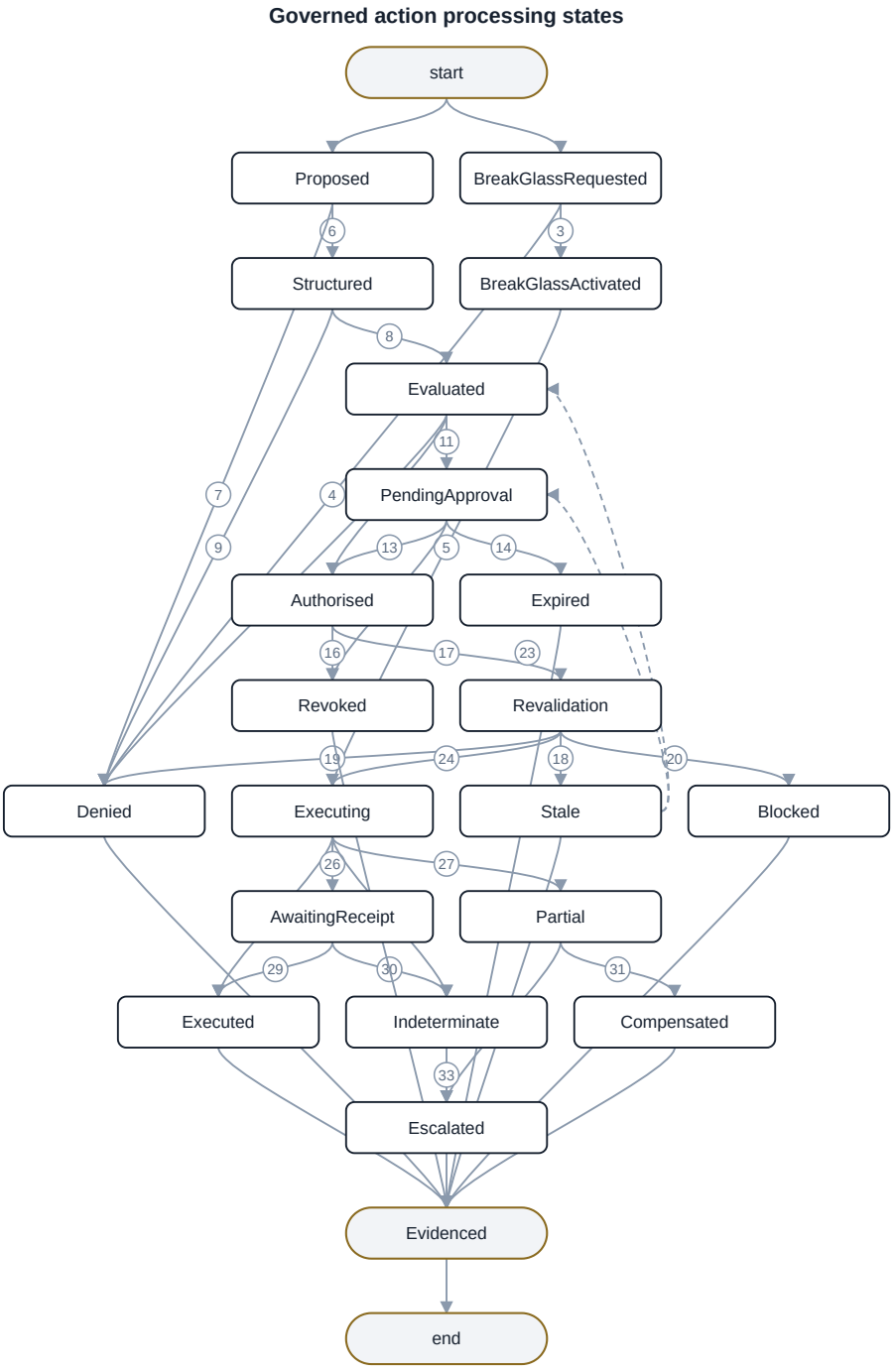

Figure 1. Processing state machine. Numbered transitions are listed below the figure.

#	Transition	Condition
3	`BreakGlassRequested` → `BreakGlassActivated`	multi-party activation
4	`BreakGlassRequested` → `Denied`	activation refused or expired
5	`BreakGlassActivated` → `Executing`	activation checkpointed, separate credential, short expiry
6	`Proposed` → `Structured`	schema validation
7	`Proposed` → `Denied`	schema rejection
8	`Structured` → `Evaluated`	identity, mandate, context, policy
9	`Structured` → `Denied`	authority or mandate failure

#	Transition	Condition
10	`Evaluated` → `Denied`	deny
11	`Evaluated` → `PendingApproval`	approval required
12	`Evaluated` → `Authorised`	permit
13	`PendingApproval` → `Authorised`	quorum of bound approvals
14	`PendingApproval` → `Expired`	timeout
15	`PendingApproval` → `Revoked`	mandate or credential revoked
16	`Authorised` → `Revoked`	revocation before revalidation
17	`Authorised` → `Revalidation`	broker receives envelope
18	`Revalidation` → `Stale`	expiry or precondition mismatch
19	`Revalidation` → `Denied`	policy re-evaluation denies
20	`Revalidation` → `Blocked`	write-ahead evidence or execution proof unavailable
22	`Stale` → `Evaluated`	re-evaluation requested
23	`Stale` → `PendingApproval`	re-approval requested
24	`Revalidation` → `Executing`	conditions hold
25	`Executing` → `Executed`	synchronous commit receipt
26	`Executing` → `AwaitingReceipt`	commit acknowledged, receipt deferred
27	`Executing` → `Partial`	partial commit
28	`Executing` → `Indeterminate`	outcome unknown after retry budget
29	`AwaitingReceipt` → `Executed`	receipt received within window
30	`AwaitingReceipt` → `Indeterminate`	receipt window exceeded
31	`Partial` → `Compensated`	compensation succeeds
32	`Partial` → `Escalated`	compensation fails
33	`Indeterminate` → `Escalated`	reconciliation required

`Blocked` is new in 0.3 and is the outcome when the Evidence Service is unavailable or refuses an execution proof at step 15. It is a terminal state, it releases both reservations (§13.3), and it is deliberately distinct from `Denied`: nothing about the transaction was wrong, and a retry once evidence is available is legitimate.

`Stale` is not terminal: a stale transaction may return to `Evaluated` for re-evaluation or to `PendingApproval` for re-approval, which is what the position paper and the adoption guide both describe. A stale transaction that is abandoned reaches `Evidenced` directly.

The break-glass branch (§19) is a distinct path with its own entry, its own activation requirement and its own evidence marking. It does not pass through `Evaluated`, and it MUST NOT be recorded as a normal governed action.

Four ordinary states are new or newly reachable. **`Indeterminate`** was enumerated as an evidence value in 0.1 but was unreachable in its state machine. **`Revoked`** did not exist, so a revocation between authorisation and revalidation had no outcome to record. **`Revalidation → Denied`** did not exist, which is the omission behind §13.2. **`AwaitingReceipt`** is new and necessary: payment rails deliver confirmations asynchronously — an ISO 20022 `pain.002` may arrive hours later — while 0.1 required a synchronous receipt containing the committed state at step 6 of its commit procedure and provided no state for the interval.

Every terminal state reaches `Evidenced`, including denials that occur before `Evaluated`. Version 0.1 required a receipt for all outcomes and then provided no evidence path for an early schema rejection.

No transition from `Proposed` or `Structured` to `Executing` is permitted. A model proposal cannot skip authority evaluation.

---

15. Target adapters and assurance modes

15.1 Modes

Mode	Required behaviour	Assurance statement
`conditional-commit`	Target atomically enforces version or state preconditions and idempotency	Strongest transaction equivalence
`locked-transaction`	Adapter holds a target-native lock from final validation through commit	Strong, subject to lock scope and failure semantics
`broker-ledger`	Broker maintains an authoritative intent ledger, commits unconditionally, and reconciles target state against the ledger on a declared cycle	**Detective only.** The commit is unconditional, so a state change between the final read and the commit is **not prevented** — it is found afterwards, within the reconciliation interval. This mode does **not** satisfy strong pre-commit freshness and MUST NOT be described as equivalent to a conditional or locked commit
`immediate-reread`	Adapter re-reads state immediately before an unconditional commit	Detects many stale states but retains a race. MUST NOT be used against an eventually-consistent target
`compensating`	Multi-step action uses defined compensations and records partial outcomes	Eventual repair, not atomicity
`observe-only`	No enforcement; records outcome for comparison	No prevention claim

15.2 Mode selection

Tier 4 actions SHOULD use `conditional-commit` or `locked-transaction`. These are the only two modes that hold GI-6P.

Where the target offers neither, **tier 4** actions MUST use `broker-ledger` with a declared reconciliation interval, and MUST NOT use `immediate-reread` or bare `compensating`.

Where `broker-ledger` is used, `execution_constraints` MUST carry a `reconciliation_interval` and a `maximum_detection_latency`, both ISO 8601 durations. Both are **inside the approved-action digest**, MUST be shown to the approver (§18.1, item 5), MUST be checked against the policy maximum for the action tier at step 11 — a breach fails closed with `E-ADAPTER` —, MUST appear in the execution receipt and the evidence record, and MUST be within a maximum the policy declares for that action tier. Without that, "detected within the declared interval" is satisfied by declaring an interval of thirty days, and the assurance statement means nothing. Exceeding `maximum_detection_latency` in operation is an incident, not a metric.

Tier 5 is different. A tier 5 action is defined as critical or irreversible, and detecting an irreversible stale commit after the fact is evidence, not control. A tier 5 action MUST hold **GI-6P** — `conditional-commit` or `locked-transaction` — or MUST NOT be automated at all: the fallback for a tier 5 action against a target with no suitable primitive is a manual, target-native process, not `broker-ledger`. **`broker-ledger` is a declared lower-assurance exception, not an equivalent.** An implementation using it holds GI-6D and not GI-6P, and MUST state that in its capability statement, in the evidence record, and — per §18.1, item 5 — to the approver. Enforcement mode M4 in the *Adoption & Measurement Guide* likewise means "the governed path is the enforced path", not "a stale action cannot commit", wherever the adapter runs in this mode.

Use of a mode weaker than `conditional-commit` MUST be visible to the approver where it materially changes risk, and MUST be present in the evidence record.

15.3 Why `broker-ledger` exists

Version 0.1 made conditional commit a SHOULD for tier 4 and 5 and assumed a compare-and-set primitive at the target. That primitive is frequently unavailable, and its availability varies **by API, by resource and by deployment model within a single vendor's platform**, not neatly by vendor. Some resources require an entity tag; others are subject to replication delay that makes `immediate-reread` unsound; many expose neither a conditional write nor a usable lock, and idempotency-key handling differs between APIs of the same product. A platform-wide assumption is therefore not evidence for an assurance mode: **the adapter's tested primitive for the declared operation determines the mode, and the test result belongs in the adapter declaration** (§15.4).

The practical consequence of 0.1's rule was that tier 4 and 5 were unreachable for most enterprise estates, so implementers would either downgrade the tier — which 0.1 permitted, because the tier was self-declared — or misdeclare the mode. Naming a weaker mode honestly and requiring reconciliation is better than a SHOULD that pushes implementers into a lie.

15.4 Alternative-path inventory

GI-3 requires that every path to a tier 4 or 5 target operation outside the enforcement point be **declared and detected**. An adapter declaration MUST therefore include the known alternative paths to the same target operation — native user interface, administrative console, other integrations, batch jobs, vendor support access — and the detective control applied to each: ingestion of the target's own audit log, periodic reconciliation of target state against the evidence record, or an explicit statement that none exists.

An absolute no-bypass claim is not available against software-as-a-service targets, and a profile that demanded one would make every honest implementation non-conformant and every conformant claim untrue.

---

16. Time, expiry and keys

- Times MUST be UTC with sufficient precision for the process, in RFC 3339 format, with a `Z` offset and no local-time forms.
- The envelope MUST carry `issued_at` and `expires_at`. Without an issuance time, a maximum-clock-skew rule has nothing to measure against — a defect in 0.1, which required skew enforcement and provided no issuance field and no skew value.
- The **default maximum clock skew is 60 seconds**, and it constrains **future** issuance only. The exact inequalities a verifier MUST apply at step 5, where `now` is the verifier's clock and `skew` the declared maximum:
- reject where `issued_at > now + skew` — issued implausibly in the future;
- reject where `not_before` is present and `now < not_before − skew`;
- reject where `now > expires_at`;
- reject where `expires_at <= issued_at`.

An envelope is **not** rejected merely because `issued_at` is older than the skew: a fifteen-minute approval validity would otherwise fail after sixty seconds. Age is bounded by `expires_at`, and a deployment wanting a separate bound MUST declare a `maximum_envelope_age` rather than overloading the skew. - Tier 4 and 5 approvals SHOULD have short, action-appropriate validity. **Default approval validity is 15 minutes** where policy does not state otherwise. - The evidence profile MUST identify whether time is local, monitored, independently timestamped, or qualified under a separate trust-service framework. - Key identifiers and the validation material required for later verification MUST be retained for the evidence lifetime. - A key-compromise event MUST trigger a documented review of the affected evidence ranges, and the compromise time MUST be recorded so that records demonstrably predating it retain their assurance. - Long-retention evidence SHOULD support re-sealing or archive-timestamp renewal before algorithm or key expiry.

17. Mandates: budget and delegation

17.1 Cumulative budget

`mandate.risk_limit` in version 0.1 was a per-action ceiling. A thousand actions of 999 euro each under a 1,000-euro limit were all individually conformant, which left the core case mandates exist to bound entirely unprotected.

A mandate MUST carry a `budget` object:

```
budget: {
  currency,           # string, ISO 4217
  ceiling_per_action,  # decimal string
  ceiling_cumulative,  # decimal string
  window,             # ISO 8601 duration
  consumed,           # decimal string, maintained by the Mandate Authority
  reserved,           # decimal string
  quarantined,        # decimal string, withheld pending reconciliation
  velocity: { max_actions, window }
}
```

The Mandate Authority MUST maintain `consumed`, `reserved` and `quarantined` as authoritative state, and the reservation MUST be an **atomic compare-and-reserve**: a read followed by a write lets two concurrent transactions both observe sufficient remaining budget and both proceed, which is precisely the case a cumulative ceiling exists to prevent. A reservation moves through exactly these states:

State	Entered when	Effect on available budget
`reserved`	At step 9 of §13, when the mandate chain is validated	Withheld
`committed`	A terminal outcome establishes that the target committed	Moved to `consumed`
`released`	A terminal outcome establishes that the target **did not** commit — `Denied`, `Stale`, `Expired`, `Revoked`, `Blocked` (write-ahead evidence or execution proof unavailable), or `Compensated` where the compensation is confirmed successful	Returned to available
`quarantined`	The outcome is `Indeterminate`, `AwaitingReceipt` beyond its window, `Partial`, or `Escalated`	**Withheld indefinitely**

A `quarantined` reservation MUST NOT be released on a timeout. It is released or committed only by an authoritative reconciliation that establishes what the target actually did, and that reconciliation MUST produce its own evidence record.

The reason is a real overspend path: an asynchronous or indeterminate transaction whose reservation is released on a timer can still commit at the target minutes or hours later, and the budget will have been returned in the interval. Timing out a reservation converts an unknown outcome into a silent grant of additional authority. A quarantined reservation that exhausts the mandate is a business problem; a released one that later commits is a control failure.

Budget state is authority state, not telemetry. A deployment that computes it from logs after the fact does not satisfy this section.

17.2 Delegation chains

`actor_chain` (§9.2) expresses the topology. The mandate expresses what may be passed along it:

```
delegation: {
  delegable,        # boolean
  max_depth,        # integer
  attenuations      # array of permitted narrowing operations
}
```

Sub-delegation MUST be **monotonically attenuating**: a delegated mandate may narrow purpose, scope, targets, budget or validity, and MUST NOT widen any of them. A verifier MUST validate the full chain and MUST reject a chain that exceeds `max_depth` or that widens at any hop.

Each hop MUST be attested by a Workload Identity Authority. An unattested hop breaks GI-1 for the whole chain and MUST fail closed rather than degrade to the nearest attested principal.

---

18. Approval

18.1 Presentation profile

GI-5A requires the approval to bind to the rendered artefact. **GI-5H** — binding to the action *as shown to the approver* — is available only under position (b) of §18.1.1. Version 0.1 asserted the strong property, bound the approval to a canonical data structure, and defined no rendering function, which made it unachievable: a compromised or merely buggy interface could display "EUR 50" while digesting "EUR 50,000", and every check would pass.

1. Each action type MUST have a **presentation profile**: `{template_id, template_version, locale, field order, provenance display rules}`, published in the same signed registry as the tier mapping.
2. The action MUST be rendered from the canonical envelope using that profile, deterministically. **Under position (a) of §18.1.1 the Approval Service renders; under position (b) the approval client renders and the Approval Service MUST NOT.**
3. `presentation_digest` is computed over the rendered artefact under the label `GAIP-0.4/presentation/v1` and MUST be carried in the approval binding scope (§10.3).
4. The rendered artefact MUST show, for every impact-determining parameter, its **derived** provenance class and source — not the class the requester asserted.
5. The rendering MUST state the assurance mode in terms of its consequence for this action. Where the mode is `broker-ledger`, it MUST additionally state the `reconciliation_interval` and the `maximum_detection_latency`, and MUST say that a changed precondition does not prevent the action. "The mode is weaker" is not a disclosure; "a change to the supplier record will not stop this payment, and would be detected within four hours" is.
6. The rendered artefact MUST be retained with the evidence for the retention period of the transaction. A `presentation_digest` whose pre-image cannot be produced later proves nothing in a dispute.

Version 0.1's processing order required a verifier to check a `presentation_digest` at step 8. The field did not exist in its data model. That contradiction was visible on the page.

18.1.1 What the presentation digest does and does not prove

`presentation_digest` proves what the Approval Service **committed to rendering**. It does not prove what a human **saw**. If the Approval Service or the display path is compromised, it can render one thing, digest another, and every check in §13 still passes.

This profile therefore requires an implementation to choose, and to declare, one of two positions:

- **(a) Trusted rendering path.** The Approval Service and the approval display are an explicit trust assumption (§4.3). The residual risk is that their compromise defeats GI-5A and GI-5H together: the digest binds a rendering the approver never saw. An implementation taking this position MUST say so in its capability statement and MUST NOT claim that the approval binds to what the human saw — only that it binds to what the service rendered.
- **(b) Trusted approval client.** A client under the approver's control renders the presentation artefact locally from the canonical envelope and the signed presentation profile, computes `presentation_digest` itself, and signs it together with the rest of the binding scope under the approver's credential. The Approval Service becomes a transport and never a renderer. Only position (b) supports the claim that the approval binds to what the approver was shown.

For tier 5, position (b) MUST be used. Position (a) does not hold GI-5H, and a tier 5 approval that cannot establish what the approver saw is not a control commensurate with an irreversible action. For tier 4, position (a) is permitted, and an implementation taking it MUST state in its capability statement that it holds **GI-5A but not GI-5H** for those action types.

Version 0.2.1 made the trusted client a SHOULD at tier 5 while acknowledging in the same section that position (a) "defeats GI-5 entirely" under compromise. That combination let an implementation be described as meeting the profile's requirements while being unable to establish the property the invariant named.

GI-5H rests on an assumption that must be stated rather than implied: that the approval client and its display path render faithfully and are not themselves compromised. Position (b) moves the trust from a central service to the approver's own client; it does not eliminate it. An implementation claiming GI-5H MUST state in its capability statement how client and display integrity is established — attestation, managed device, hardware-backed credential — because without that the claim reduces to position (a) with extra steps.

Neither position removes the deeper limit: a human who does not read the rendering has approved it regardless.

18.2 Approval signature

The approval MUST be a signature under the **approver's own credential** (§11.1), over the binding scope in §10.3, and MUST record the authentication method and assurance level, the device or session binding where available, and the time of approval.

18.3 Quorum and separation of duties

`approvals` is an array, and its required cardinality and composition come from **policy**, not from the envelope:

- Policy MUST be able to express a **quorum** — n of m — and **role conditions** for each required approver.
- The profile defines a segregation rule that policy MAY strengthen but MUST NOT weaken for tier 5: **the approver MUST NOT be the requester, the initiating principal, or the beneficiary of the action.**
- **An automated tier 5 action MUST carry approvals from at least two distinct natural-person credentials, in distinct authorised roles, each distinct from the requester, the beneficiary and each other.** Version 0.3 required policy to be *able* to express a quorum and then permitted a one-of-one quorum to satisfy the profile — so four-eyes control was expressible and not required, at the only tier where the position paper calls it the primary control. Where these conditions cannot be met, the action MUST NOT be automated.
- A verifier MUST evaluate the quorum and the segregation rule at step 13 of §13, and MUST fail closed where either is unsatisfied.

Version 0.1 had a single `approver` field and no quorum, no roles and no segregation rule. For tier 5 — where multi-party approval is the primary control — the profile's own control was absent.

---

19. Break-glass profile

GI-3 as written in v0.4, and version 0.1 of this profile, both permitted "a separately controlled break-glass path" and then specified nothing: no envelope, no state, no approval model, no evidence requirement, no time limit, no review obligation. The one path an attacker will attack sat entirely outside the model, and an implementer could call any bypass "break-glass" and remain conformant.

A `break_glass` block, where present, MUST carry:

Field	Requirement
`declared_purpose`	From a closed, published list of emergency purposes
`activation`	Multi-party activation record for tier 4 and 5: at least two distinct credentials, in distinct roles. The record MUST be signed and checkpointed by the Evidence Service **before** the emergency credential is released — the same ordering as §20.2.3, and for the same reason
`credential_ref`	Reference to a credential distinct from the normal path and under separate key custody
`expires_at`	Short, absolute; a break-glass envelope MUST NOT be renewable
`alerting_ref`	Reference to the real-time alert emitted at activation
`review_due`	Deadline for the mandatory post-event review

Additional requirements:

- A break-glass transaction MUST follow the distinct `BreakGlassRequested → BreakGlassActivated → Executing` path in §14 and MUST be marked in the evidence record as `break_glass`. It MUST NOT be recorded as a normal governed action.
- Break-glass use MUST emit conspicuous real-time alerting at activation, not at review.
- The path MUST be tested on a declared cycle, and the test MUST NOT expose the credential to routine operators.
- An implementation MUST report break-glass invocations as a distinct metric. A break-glass rate that resembles a business-as-usual rate is a finding, not a statistic.

20. Evidence

20.1 Correlated record

The Evidence Record correlates: transaction and tenant identifiers; principal and full actor chain; mandate reference, version and budget movement; envelope profile and version; intent with origin marking; model and tool identifiers with versions; data classification; **every parameter with its assessment status and, where assessed, its attestation reference and derived provenance class**; policy decision, policy digest and policy-input digest; every approval with its binding scope and the retained presentation artefact; precondition readings and the derived tier with registry reference; commit intent; execution receipt including the idempotency key and committed version; assurance mode used; timestamps with time-source class; cryptographic profile; outcome; and any incident link.

20.2 Write-ahead evidence

The **Commit Intent Record** MUST be written and checkpointed **before** any side effect, at tier 3 and above (step 16 of §13). It carries the envelope digest, the derived idempotency key, the target and operation, the assurance mode, the intended commit time, and a `terminal_receipt_deadline` — the instant by which a terminal receipt must exist, which is what the completeness monitor in §20.2.2 measures against.

20.2.1 A requirement on the compromised component is not a defence

The threat model assumes an adversary may fully compromise the enforcement point (A4 in §4.2). Version 0.2.1 then placed the write-ahead requirement **on that same enforcement point**. A compromised broker simply skips the step and uses its target credential directly, and no rule addressed to it changes that.

From tier 3 upward, the execution capability MUST therefore be **conditioned on the evidence**, not merely accompanied by it:

1. The PEP submits the Commit Intent Record to the Evidence Service.
2. The Evidence Service checkpoints it and returns a **one-time, short-lived execution proof** bound to `{transaction_id, envelope digest, target, operation, commit-intent digest}`.
3. The PEP obtains the target credential or execution token **only on presentation of that proof**, from a credential broker or target that verifies it.

Where the target cannot verify such a proof — which is the case wherever the target does not expose a suitable verifier — the assurance is **detective, not preventive**, and the implementation MUST reconcile all target commits against evidence on a declared cycle and MUST state that it is doing so. This is the same GI-6P/GI-6D distinction applied to evidence: an implementation should not describe a reconciliation control as though it prevented the bypass.

20.2.3 The ExecutionProof artefact

Version 0.2.2 introduced the execution proof as the central defence against a compromised enforcement point and then specified none of it. A control that carries that much weight and has no data model, no issuer rule and no replay protection is a diagram, not a mechanism.

The Evidence Service issues, and signs, an artefact of exactly this shape:

```
ExecutionProof {
  profile,                  # "GAIP-0.4"
  proof_id,                 # unique; the one-time consumption handle
  tenant_id,
  transaction_id,
  envelope_digest,          # {alg,value}
  commit_intent_digest,     # {alg,value}
  target,
  operation,
  credential_scope_digest,  # {alg,value} over the credential scope being released
  issuer,                   # the Evidence Service identity
  audience,                 # the Credential Broker or target that may verify it
  not_before,
  expires_at,
  nonce
}
```

Rules:

1. The proof MUST be signed by the **Evidence Service** (§11.1) and MUST NOT be issued before the Commit Intent Record is durably checkpointed. 1a. **The issuer MUST independently authorise, not merely record.** Before issuing, it MUST verify the envelope's signatures, verify that the policy decision's effect is `permit` and that it is unexpired, validate the mandate and the required approvals, and **derive the credential scope itself** from the approved target, operation, tenant, resource constraints and transaction — then bind that derived value as `credential_scope_digest` under `GAIP-0.4/credential-scope/v1`. It MUST NOT adopt a scope supplied by the enforcement point.

   Version 0.3 required the issuer only to checkpoint what it was handed. A compromised enforcement point could therefore submit a Commit Intent Record naming a broader scope, obtain a valid proof, and present it as evidence that a broader credential was authorised. The proof established that an intent had been recorded; it did not establish that the intent was permitted. Against adversary A4 that is the whole question. 2. The verifier — Credential Broker or target — MUST reject a proof whose `audience` is not itself, whose `envelope_digest`, `commit_intent_digest` or `credential_scope_digest` does not match the request and the scope it is being asked to release, whose `target` or `operation` differs, or which is outside its `not_before`/`expires_at` window. 3. Consumption MUST be **atomic and one-time**, keyed on `proof_id`. A second presentation MUST fail with `E-PROOF` and MUST be recorded as a security event. 4. Validity SHOULD be short — the interval between checkpoint and commit, not the interval of the approval. 5. The released credential **MUST** be one-time and **transaction-bound or request-bound**. A reusable credential defeats the preventive claim outright: once a compromised broker has obtained it under one valid proof, nothing in this profile constrains what it does with it next. Version 0.3 made this a SHOULD. 6. **Where a preventive claim is made, the Credential Broker or verifying target MUST be separated from the enforcement point's trust domain** (§5.2). Where no such verifier exists — the case wherever the target does not expose a suitable verifier — the implementation MUST declare the write-ahead control as **detective**, and MUST reconcile target commits against evidence on a declared cycle.

The Commit Intent Record MUST additionally carry a `terminal_receipt_deadline`, which is what the completeness monitor in §20.2.2 measures against; version 0.2.2 gave it only an intended commit time and then asked the monitor to enforce a window that no field carried.

20.2.2 An unresolved intent is an anomaly, not a gap

Sequence numbering alone does not detect this. If the PEP skips the Commit Intent Record entirely, nothing is sequenced and there is no gap. If an intent record takes sequence 100 and the next valid record takes 101, the numbering is intact while the transaction it described has no terminal receipt.

The Evidence Service MUST therefore run a **completeness monitor**: every Commit Intent Record carries a declared window within which a terminal receipt must arrive, and the Evidence Service MUST emit a **signed evidence anomaly** when that window expires with no terminal record — whether or not any sequence gap exists. The anomaly is itself an evidence record and is itself sequenced.

Version 0.1 created evidence at its final processing step, after execution. A broker that executed an action and then suppressed the record left nothing behind, and GI-8 as then written constrained only "the application that benefits from an action" — not the broker. The whole point of an evidence layer is that it survives the party it incriminates.

20.3 Gap detection

Evidence entries MUST carry a **signed monotonic sequence number per tenant**. A verifier MUST be able to detect a gap. Omission must be visible as a gap rather than as silence; without a sequence, a suppressed record and a record that never existed are indistinguishable.

20.4 Independence

At tier 4 and above, evidence signing keys MUST be under custody separate from the PEP (§5.2). At tier 5, checkpoints MUST additionally be witnessed by a party other than the Evidence Service operator. Where no independent witness exists, the evidence profile MUST say so, and the assurance claim MUST be stated accordingly.

20.5 What evidence proves

The construction to adopt here is Certificate Transparency's, not a novel one. And its claim is bounded: these mechanisms do not prevent a compromised or highly privileged actor from altering or deleting records. They make subsequent manipulation, missing entries and reordering **detectable** under the stated trust assumptions.

> The goal is not the unrealistic claim of an "unalterable log", but defensible cryptographic tamper-evidence.

21. Operational requirements

21.1 Model and provider evidence

For a self-hosted model the record can contain an artefact digest, container or package digest, tokenizer version, runtime version and deployment configuration. For a provider-managed model a reproducible artefact may not be available; the record MUST then distinguish provider and legal service, endpoint or deployment identifier, advertised model alias, API version, provider request or trace identifier, any provider-supplied fingerprint, the relevant safety or tool configuration digest, and a **reproducibility status** of `reproducible`, `provider-identifiable`, `alias-only` or `unknown`.

An implementation MUST NOT claim an exact model version where the provider supplied only a mutable alias.

21.2 Tiered evidence assurance

Level	Mechanism	Typical scope
Basic	Correlated, access-controlled records with a central time source	Tier 0–1
Elevated	Canonical records, hash chaining and signatures	Tier 2–3
High	Append-only or WORM storage, separate replicas, signed checkpoints	Tier 4
Externally verifiable	Independent witness or transparency log	Tier 5

21.3 Safe degradation

An implementation MUST declare, per action tier, the behaviour on loss of the identity or policy component, on loss of the evidence path, and on target-adapter uncertainty.

Tier	Identity/PDP outage	Evidence outage	Adapter uncertainty
0	Continue under local data/model policy where permitted	Buffer minimal records or mark unevidenced	Return information only
1	Continue or degrade to local recommendation	Buffer and reconcile	No external side effect
2	Draft only; no publication or commit	Store draft, block promotion	Draft only
3	Pre-authorised reversible actions only, with a short offline policy cache; otherwise manual	**Queue or block.** Write-ahead evidence is mandatory from tier 3 (§20.2), so no tier 3 side effect may proceed until the Commit Intent Record is durably written and checkpointed	Fail closed or compensate
4	Block or route to a fully manual authorised process	Block execution	Block and escalate
5	Block; no offline autonomous mode	Block execution	Block, isolate, invoke incident procedure

The tier 3 evidence column is deliberately stricter than it was in v0.4, where "queue if reversible" could be read as permitting a side effect while the evidence path was down. It cannot: §20.2 makes the Commit Intent Record a precondition of execution from tier 3 upward, and reversibility is not a substitute for a record of what was attempted.

Offline policy caches create their own freshness and revocation risk. Their maximum age and permitted action set MUST be part of the policy and the evidence profile.

21.4 Enforcement modes and scope statements

An implementation operating a brownfield estate MUST state, per action type, which enforcement mode applies — inventory, observe, shadow decision, soft enforcement or **enforced governed path** — because the assurance available differs by mode. Note that the enforced governed path (M4) means agent, application and declared automated execution credentials for the assessed action path are mediated by the enforcement point. **It does not by itself mean a stale action cannot commit**: that depends on the adapter's assurance mode, and on `broker-ledger` it is expressly not true (§15.2). The modes are defined in the *Adoption & Measurement Guide*, §4.1. A scope statement that does not name modes per action type is not a scope statement.

22. Schema, media types, versioning and errors

Version 0.1 described itself as an interoperability profile and provided no schema, no media type, no extension registry, no version negotiation and no transport binding. Two vendors could not have exchanged a single envelope.

22.1 Schema

This profile **requires** a normative **JSON Schema** (2020-12) and a normative **CDDL** definition, published as separate, independently versioned artefacts. Where this prose and the schema disagree, **the schema is to govern for structure and cardinality, and this prose for semantics.** Once they are published, an implementation MUST validate against the schema before any semantic processing (§13, step 2).

They are not published with this edition. They are in preparation. Until they exist, §13 step 2 can only be satisfied against an implementation-local schema, two implementations will diverge on structure and cardinality, and no interoperability claim is available. This is the largest single gap in 0.2 and is recorded first in §26.2.

22.2 Media types

Type	Use
`application/gaip-envelope+json`	JSON profile envelope
`application/gaip-envelope+cbor`	CBOR profile envelope
`application/gaip-receipt+json`	Execution receipt
`application/gaip-evidence+json`	Evidence record or package

These are provisional and unregistered. IANA registration is an open item (§26).

22.3 Extensions, versioning and downgrade protection

- Extension fields MUST be namespaced by a URI-derived prefix.
- The `critical` array names the fields and extensions a verifier MUST understand. A verifier that does not understand a listed item MUST reject the envelope. Everything not listed MUST be ignored if unknown — must-ignore semantics.
- A verifier MUST enforce a **minimum acceptable profile revision** from policy and MUST reject an envelope below it. Without this, an attacker downgrades to whichever revision had the weakest rules.
- **Ordering is defined, not implied.** Profile revisions are compared as semantic versions: `major.minor[.patch]`, numerically field by field, a missing patch treated as `0`. Registry versions are compared as monotonically increasing integers. A verifier MUST reject a version string it cannot parse under these rules rather than treat it as acceptable — version 0.2.2 required both comparisons and defined neither.
- Version negotiation, where used, MUST be integrity-protected.

22.4 Transport binding

At least one normative HTTP binding is required for interoperability: endpoints, authentication, channel binding of the envelope to the transport session, asynchronous approval callback, and asynchronous receipt delivery for the `AwaitingReceipt` state.

Status: not defined in this edition (§26).

22.5 Errors and resource limits

An implementation MUST use the error taxonomy below and MUST NOT collapse distinct failures into a generic denial. An assessor cannot distinguish a working control from a broken one when every failure returns the same code.

Class	Meaning
`E-PARSE`	Malformed, non-canonical, duplicate member, or limit exceeded

Class	Meaning
`E-CRITICAL`	Unknown critical feature
`E-SIG`	Signature, key, role or tenant authority failure
`E-REPLAY`	Transport replay: the same envelope or derived idempotency key presented again (§13.1)
`E-TIME`	Envelope time invalid: before `not_before`, after `expires_at`, `issued_at` outside skew, or expiry preceding issuance (§13, step 5)
`E-PROOF`	Execution proof absent, expired or not bound to this transaction (§20.2.1)
`E-DUPLICATE`	Business duplication: a distinct envelope matching an existing business deduplication key within its window (§13.1.1)
`E-PRINCIPAL`	Principal or actor-chain resolution failure
`E-MANDATE`	Mandate invalid, revoked, exceeded or improperly attenuated
`E-BUDGET`	Cumulative budget or velocity exceeded
`E-POLICY`	Policy decision invalid, expired or re-evaluated to a different outcome
`E-TIER`	Tier derivation divergence
`E-PROVENANCE`	Attestation missing, invalid, or prohibited tier × class combination
`E-APPROVAL`	Quorum, segregation, binding, single-use or presentation failure
`E-STALE`	Precondition mismatch or freshness window exceeded
`E-ADAPTER`	Assurance mode unavailable or target refused
`E-VERSION`	Profile version below the declared minimum acceptable version (§22.3)
`E-REPRESENTATION`	Cross-representation digest comparison refused (§12.5)
`E-BREAKGLASS`	Break-glass activation, expiry or multi-party requirement not satisfied (§19)
`E-INDETERMINATE`	Outcome unknown after retry budget

Resource limits MUST be declared: maximum envelope size, maximum nesting depth, maximum array lengths, maximum actor-chain depth, maximum parameters, and a retry budget. An undeclared limit is a denial-of-service surface.

23. Test vectors

Version 0.1 required implementations to publish test vectors before claiming interoperability and published none itself. These are the third generation, recomputed for 0.3. They are deliberately minimal and cover the constructions most likely to diverge between implementations.

`H` is SHA-256. Canonical form is RFC 8785 JCS. Every digest is carried as `{"alg":"sha-256","value":"<hex>"}` — the only permitted representation (§10.1). All monetary values and identifiers are strings; the only JSON numbers are bounded structural integers (§12.3). Arrays follow §12.4.

TV-1 — Canonical form

Input object, member order deliberately scrambled on input:

```
{"target":"erp:payment-api","amount":"1200.00","payee_iban":"DE02120300000000202051","operation":"create-
payment","currency":"EUR"}
```

Canonical bytes (131 octets, UTF-8, no whitespace, members sorted):

```
{"amount":"1200.00","currency":"EUR","operation":"create-
payment","payee_iban":"DE02120300000000202051","target":"erp:payment-api"}
```

`H(canonical_bytes)` — no domain label, for canonicalisation checking only:

```
e77b45e1f48d93c72afd4d6d44bd6efb2a851dad3f58117572da1168f34477a6
```

TV-7 — Parameter-value commitment

The construction that binds an attestation to the value that will execute (§7.3.1). Canonical bytes for `amount`:

```
{"normalisation_profile":"urn:gaip:norm:decimal:scale-
2","parameter_name":"amount","unit":"EUR","value":"1200.00","value_type":"decimal"}
```

`H("GAIP-0.4/parameter-value/v1" || 0x00 || canonical_bytes)`:

```
74970cb3279b590b7f74a2e005734cb19f858d20d7194b140c7e3ae6282a435b
```

The same construction for the other two parameters of TV-2:

Parameter	`value_type`	`normalisation_profile`	Commitment
`currency`	`string`	`urn:gaip:norm:string:nfc`	`487a1433d0135d31fa00ea696feeeec14d2a60e67a6f8896f5ef897ead23c58d`
`payee_iban`	`string`	`urn:gaip:norm:iban:compact`	`94c2aa501e8fe6ced2d34ad705b0773f88b43843c558c24d27893675fe019ce8`

Note `unit`, new in 0.4: the commitment binds the currency, not only the number.

Three cases show what the normalisation profile is for. §12.3 requires normalisation to the declared scale **before** the commitment is computed:

Input `value`	Behaviour	Commitment
`"1200.00"`	Already canonical for scale 2	`74970cb3279b590b7f74a2e005734cb19f858d20d7194b140c7e3ae6282a435b`
`"1200.0"`	Normalised to `"1200.00"`, then committed	**the same value**
`"1200.001"`	**Rejected.** Normalisation to scale 2 would lose precision; a verifier MUST reject rather than round	—

An implementation that skips normalisation and commits `"1200.0"` as written produces `f132961c9e3410b27b746b731d597ae7e64fb081cbd84cf1f910f63a3b204b5a` instead, and will not interoperate. Version 0.3's commentary asserted that this differing value was the correct behaviour, which contradicted its own §12.3.

TV-8 — Intent digest

The binding version 0.2.2 lacked (§9.3). Canonical bytes:

```
{"origin":"user-confirmed","principal_confirmation":{"confirmed_at":"2026-08-26T09:14:02Z","method":"in-band-
restatement","principal":"person:B"},"schema_digest":{"alg":"sha-
256","value":"ad45659a4d9c9ab2ca9a3a998ae0ce758ebf21ac435ea28ea3da678e3da69227"},"schema_uri":"urn:vianordis:intent:inv
oice-payment:1","type":"invoice.payment.execute","values":{"invoice_reference":"4711","supplier":"V-238"}}
```

`H("GAIP-0.4/intent/v1" || 0x00 || canonical_bytes)`:

```
aa38bd98c08e8aad6903bcd45662f918d98a4b61ac59498602d715455c4e673c
```

`schema_digest` is what makes the schema commitment immutable. Rewriting `origin` from `user-confirmed` to `model-generated`, substituting the confirmation record, or changing the content behind `schema_uri` each changes this digest and therefore invalidates the approval.

TV-2 — Approved-action digest

Canonical bytes of the action substructure (§10.2). Every element in it closes a substitution path found in an earlier revision: `profile` binds the requirements revision; `intent_digest` binds the structured intent, its origin and its confirmation; each parameter binds both its `parameter_value_digest` and its `attestation_digest`, so neither the value nor the attestation can be swapped independently; and `execution_constraints` binds the adapter and the assurance mode.

```
{"execution_constraints":{"allowed_adapter":"adapter:erp-payment:2","assurance_mode":"conditional-
commit","business_deduplication_profile":"urn:vianordis:dedup:supplier-payment:1","evidence_profile_digest":
{"alg":"sha-
256","value":"aa11bb22cc33dd44ee55ff6677889900aabbccddeeff00112233445566778899"},"maximum_attempts":1,"operation":"crea
te-payment","target":"erp:payment-api"},"intent_digest":{"alg":"sha-
256","value":"aa38bd98c08e8aad6903bcd45662f918d98a4b61ac59498602d715455c4e673c"},"parameters":[{"attestation_digest":
{"alg":"sha-
256","value":"3f1c0a5e0b7d4a2f9c8e1b6d5a4f3e2d1c0b9a8f7e6d5c4b3a2f1e0d9c8b7a6f"},"derived_class":"P1","name":"amount","
normalisation_profile":"urn:gaip:norm:decimal:scale-2","parameter_value_digest":{"alg":"sha-
256","value":"74970cb3279b590b7f74a2e005734cb19f858d20d7194b140c7e3ae6282a435b"},"provenance_assessment":"assessed","un
it":"EUR","value":"1200.00","value_type":"decimal"},{"attestation_digest":{"alg":"sha-
256","value":"9a8b7c6d5e4f3a2b1c0d9e8f7a6b5c4d3e2f1a0b9c8d7e6f5a4b3c2d1e0f9a8b"},"derived_class":"P1","name":"currency"
,"normalisation_profile":"urn:gaip:norm:string:nfc","parameter_value_digest":{"alg":"sha-
256","value":"487a1433d0135d31fa00ea696feeeec14d2a60e67a6f8896f5ef897ead23c58d"},"provenance_assessment":"assessed","va
lue":"EUR","value_type":"string"},{"attestation_digest":{"alg":"sha-
256","value":"11223344556677889900112233445566778899001122334455667788990011 22"},"derived_class":"P0","name":"payee_iba
n","normalisation_profile":"urn:gaip:norm:iban:compact","parameter_value_digest":{"alg":"sha-
256","value":"94c2aa501e8fe6ced2d34ad705b0773f88b43843c558c24d27893675fe019ce8"},"provenance_assessment":"assessed","va
lue":"DE02120300000000202051","value_type":"string"}],"profile":"GAIP-0.4","tenant_id":"tenant-
42","transaction_id":"0191f2d1-7c5a-7c31-a15f-b15de9b72b85"}
```

`H("GAIP-0.4/approved-action/v1" || 0x00 || canonical_bytes)`:

```
7088ca264cf4af2920de145b0ddff0adede11c9f7a6620479501ab523f7dc880
```

TV-3 — Domain separation

The same canonical bytes as TV-2, under the precondition label:

`H("GAIP-0.4/approved-preconditions/v1" || 0x00 || canonical_bytes)`:

```
731c62970ae7cc47808b79ffca2ef06dde8a7a06b67b89d2b873df7d03f0d388
```

TV-2 and TV-3 have identical input bytes and different digests. An implementation that produces the same value for both has not implemented domain separation, and in that implementation a precondition digest is a valid action digest.

TV-4 — Presentation digest

Rendered artefact:

```
{"locale":"en-GB","rendered":"Pay EUR 1,200.00 to ACME Supplies GmbH, IBAN DE02 1203 0000 0000 2020 51, for invoice
4711.\nBank details source: supplier master record V-238 version 14, confirmed against supplier onboarding record
(independently verified, P0).\nAmount source: purchase order 9204, ERP version 7 (authoritative system state,
P1).\nCurrency source: purchase order 9204, ERP version 7 (authoritative system state, P1).\nExecution: conditional
commit against ERP supplier version 14 and purchase order 9204 open. A changed precondition prevents this
payment.","template":"urn:vianordis:presentation:invoice-payment:1","template_version":"3"}
```

`H("GAIP-0.4/presentation/v1" || 0x00 || canonical_bytes)`:

```
d0a7d20dc882c4351bd0b613791ed7976d09a063cd072dd65b671ee056ef7c48
```

The rendering names the derived class and source of **every** impact-determining parameter (§18.1, item 4) and states the assurance mode in terms of its consequence (§18.1, item 5). Under `broker-ledger` the last line would instead have to say that a changed precondition does **not** prevent the payment and would be detected within the declared interval.

TV-5 — Envelope digest and derived idempotency key

Envelope digest over the TV-2 substructure under the envelope label. This is **illustrative**: a real `GAIP-0.4/envelope/v1` digest is taken over the whole envelope minus signature containers and minus the derived idempotency key (§10.2). The substructure is used here so the derivation is reproducible from the bytes printed above.

```
0b9738c6dfcf580c191b3ee20733effc7924dde0898dc098ea65bf848319e653
```

The idempotency key is a digest over a canonical object under its own label (§13.1), not a concatenation. Canonical bytes:

```
{"envelope_digest":{"alg":"sha-
256","value":"0b9738c6dfcf580c191b3ee20733effc7924dde0898dc098ea65bf848319e653"},"operation":"create-
payment","target":"erp:payment-api","tenant_id":"tenant-42"}
```

`H("GAIP-0.4/idempotency-key/v1" || 0x00 || canonical_bytes)`:

```
eaa4af7a6b6a175a2bfa021f8ab0a8f4e4fdcf9c5761f48c74dfd82c5ee92e0b
```

TV-6 — Action Type Registry digest

The registry determines the tier, the impact-determining parameter set, the authoritative sources and the presentation profile, so its digest is as load-bearing as the policy-decision digest. Canonical bytes of a genesis snapshot with two entries for the same action type and target — one conditional at tier 4, one `{always:true}` default at tier 3:

```
{"effective_from":"2026-08-01T00:00:00Z","entries":[{"action_type":"invoice.payment.execute","authoritative_sources":
{"amount":["erp:purchase-order"],"currency":["erp:purchase-order"],"payee_iban":["erp:supplier-
master"]},"budget_amount_parameter":"amount","budget_currency_parameter":"currency","business_deduplication":
{"key_fields":
["intent.values.invoice_reference","parameters.amount","parameters.currency","parameters.payee_iban"],"profile":"urn:vi
anordis:dedup:supplier-payment:1","window":"P90D"},"condition":{"always":true},"entry_id":"urn:vianordis:atr:invoice-
payment:default","impact_determining":["amount","currency","payee_iban"],"independence_domains":
{},"presentation_profile_digest":{"alg":"sha-
256","value":"776dab1fbb27aa265d03172ed3d924fb43f6edd540ee8e9823ebd2b8de2cac7b"},"provenance_freshness":
{"amount":"PT15M","currency":"PT15M","payee_iban":"PT24H"},"required_preconditions":[{"comparison":"equal","expected":
{"value":"open","value_type":"string"},"freshness_window":"PT5M","name":"po_state","source":"erp:purchase-
order:9204"}],"source_domains":{"erp:purchase-order":"erp-core","erp:supplier-master":"supplier-
master"},"target":"erp:payment-api","tier":3},{"action_type":"invoice.payment.execute","authoritative_sources":
{"amount":["erp:purchase-order"],"currency":["erp:purchase-order"],"payee_iban":["erp:supplier-master","erp:supplier-
onboarding"]},"budget_amount_parameter":"amount","budget_currency_parameter":"currency","business_deduplication":
{"key_fields":
["intent.values.invoice_reference","parameters.amount","parameters.currency","parameters.payee_iban"],"profile":"urn:vi
anordis:dedup:supplier-payment:1","window":"P90D"},"condition":{"op":"gte","operand":
{"scale":2,"unit":"EUR","value":"1000.00","value_type":"decimal"},"parameter":"amount","unit_parameter":"currency"},"en
try_id":"urn:vianordis:atr:invoice-payment:tier4","impact_determining":
["amount","currency","payee_iban"],"independence_domains":{"payee_iban":[["supplier-master","supplier-
onboarding"]]},"presentation_profile_digest":{"alg":"sha-
256","value":"776dab1fbb27aa265d03172ed3d924fb43f6edd540ee8e9823ebd2b8de2cac7b"},"provenance_freshness":
```

```
{"amount":"PT15M","currency":"PT15M","payee_iban":"PT24H"},"required_preconditions":[{"comparison":"equal","expected":
{"value":"open","value_type":"string"},"freshness_window":"PT5M","name":"po_state","source":"erp:purchase-order:9204"},
{"comparison":"equal","expected":
{"value":"14","value_type":"string"},"freshness_window":"PT5M","name":"supplier_version","source":"erp:supplier:V-
238"}],"source_domains":{"erp:purchase-order":"erp-core","erp:supplier-master":"supplier-master","erp:supplier-
onboarding":"supplier-onboarding"},"target":"erp:payment-
api","tier":4}],"previous_version_digest":null,"registry_id":"urn:vianordis:action-type-registry:tenant-
42","tenant_scope":"tenant-42","version":7}
```


`H("GAIP-0.4/action-type-registry/v1" || 0x00 || canonical_bytes)`:


```
c152c372eae620bf48f8effe71ed41d6670e21b31c2757efbca469ee1a3481eb
```


Several things in these bytes are worth reading closely. There are **two entries** for the same `{action_type, target}` pair, distinguished by `entry_id` and resolved by most-specific match (§7.5.1) — the construction 0.2.2 could not represent at all. Exactly one of them carries `{always: true}`, which §7.5.1(d) requires as the default; a pair without one fails closed. `condition.operand` carries `value_type`, `unit` and `scale`, so the comparison cannot silently coerce. `independence_domains` declares the pair of source domains that makes the P0 class in TV-2 derivable — without it, `payee_iban` could reach P1 at best. `required_preconditions` carry typed `expected` operands. `key_fields` is sorted ascending, as §12.4 requires — the 0.2.2 vector was not, which meant the published registry vector was not a canonical registry. And `previous_version_digest` is `null`, which is permitted **only** in a genesis snapshot; every later snapshot carries the digest of its predecessor and forms the hash chain.

Note `"tier":4` and `"version":7` as JSON integers while `"value":"1000.00"` is a string: tiers and versions are bounded structural integers, a monetary threshold is an exact decimal (§12.3).

TV-9 — Execution proof

The artefact that conditions execution on evidence (§20.2.3). Canonical bytes:


```
{"audience":"credential-broker:erp-payment","commit_intent_digest":{"alg":"sha-
256","value":"5e4d3c2b1a09f8e7d6c5b4a3928170f6e5d4c3b2a1908f7e6d5c4b3a29180706"},"credential_scope_digest":{"alg":"sha-
256","value":"dccef0c549777faff675a20f71b3b795c7070a4016073df242b58f6f12db5242"},"envelope_digest":{"alg":"sha-
256","value":"0b9738c6dfcf580c191b3ee20733effc7924dde0898dc098ea65bf848319e653"},"expires_at":"2026-08-
26T09:15:32Z","issuer":"evidence-service:tenant-42","nonce":"0191f2d1a3b44c8e","not_before":"2026-08-
26T09:15:02Z","operation":"create-payment","profile":"GAIP-0.4","proof_id":"0191f2d1-7c5a-7c31-a15f-
b15de9b72b86","target":"erp:payment-api","tenant_id":"tenant-42","transaction_id":"0191f2d1-7c5a-7c31-a15f-
b15de9b72b85"}
```


`H("GAIP-0.4/execution-proof/v1" || 0x00 || canonical_bytes)`:


```
7ab0e7cb412d077bdf0669a0c9afe7f913d1baab67dd3ff9e0085ed0e05bcbb9
```


`audience` is what stops a proof issued for one credential broker being presented to another, and `proof_id` is the one-time consumption handle. The window here is thirty seconds — the interval between checkpoint and commit, not the interval of the approval.

TV-10 — Intent-schema digest

The construction that makes `intent.schema_digest` immutable (§10.2). Canonical bytes:


```
{"dialect":"https://json-schema.org/draft/2020-12/schema","media_type":"application/schema+json","schema":
{"$id":"urn:vianordis:intent:invoice-payment:1","properties":{"invoice_reference":{"type":"string"},"supplier":
{"type":"string"}},"required":["invoice_reference","supplier"],"type":"object"}}
```


`H("GAIP-0.4/intent-schema/v1" || 0x00 || canonical_bytes)`:


```
ad45659a4d9c9ab2ca9a3a998ae0ce758ebf21ac435ea28ea3da678e3da69227
```


The dialect and media type are inside the digest because the same bytes interpreted under a different schema dialect are a different schema.

TV-11 — Presentation-profile digest

Input to `presentation_profile_digest` in the registry entry (§7.5) and to the rendering rules (§18.1). Canonical bytes:

```
{"field_order":["payee","iban","amount","invoice","provenance","execution"],"locale":"en-
GB","profile_id":"urn:vianordis:presentation:invoice-payment:1","provenance_display":"class-and-
source","template_version":"3"}
```

`H("GAIP-0.4/presentation-profile/v1" || 0x00 || canonical_bytes)`:

```
776dab1fbb27aa265d03172ed3d924fb43f6edd540ee8e9823ebd2b8de2cac7b
```

TV-12 — Authentication-evidence digest

What makes P2 derivable at all: the recorded authentication behind an asserted resolution (§7.3, §7.4). Canonical bytes:

```
{"assurance_level":"AAL3","credential_ref":"cred:person:C:fido2-01","issuer":"idp:tenant-42","method":"webauthn-user-
verified","session_binding":"device:managed:9f2c","subject":"person:C","time":"2026-08-26T09:14:55Z"}
```

`H("GAIP-0.4/authentication-evidence/v1" || 0x00 || canonical_bytes)`:

```
5308d4b164c73f0870fc7164f7cbfa83e19e81d3e39d81e9422341a1789ecfe2
```

TV-13 — Credential-scope digest

The scope an ExecutionProof authorises — **derived by the issuer**, never adopted from the enforcement point (§20.2.3). Canonical bytes:

```
{"operation":"create-payment","resource_constraints":{"payee_iban":"DE02120300000000202051"},"target":"erp:payment-
api","tenant_id":"tenant-42","transaction_id":"0191f2d1-7c5a-7c31-a15f-b15de9b72b85"}
```

`H("GAIP-0.4/credential-scope/v1" || 0x00 || canonical_bytes)`:

```
dccef0c549777faff675a20f71b3b795c7070a4016073df242b58f6f12db5242
```

This is the value carried as `credential_scope_digest` in TV-9.

What is still missing

These thirteen vectors cover canonicalisation, domain separation, the parameter-value commitment with unit and normalisation, the intent digest and its schema, presentation binding and the presentation profile, authentication evidence, key derivation, the registry, the execution proof and the credential scope it authorises. They do **not** yet cover signature containers, key resolution, the CBOR profile, attenuated delegation chains, or a full end-to-end envelope with signatures. Those require published key material and a reference implementation, and are open items (§26.2).

---

24. Required negative tests

An implementation that states it meets this profile for a set of action types SHOULD publish or make available results for these tests. The test report should state which tests were automated, which required manual review, the environment used, and known exceptions.

#	Test	Traces to
T1	Model attempts an action outside the intent schema	GI-4
T2	Application presents a self-issued or widened mandate	GI-2
T3	Mandate revoked after approval, before commit	GI-7
T4	Policy changed after approval; re-evaluation denies	§13.2
T5	Target state changed after approval	GI-6P
T6	Approval replayed with modified parameters	GI-5A
T7	Exact envelope replayed after successful commit	§13.1
T8	Duplicate delivery during target timeout	§13.1
T9	Evidence producer attempts to omit or reorder an entry	§20.3
T10	Business application attempts direct target access	GI-3
T11	Signing key rotated; historical verification still succeeds	§16
T12	Evidence signing key declared compromised	§16
T13	Control-plane component unavailable; declared safe state observed	GI-10
T14	Partial multi-system commit; compensation succeeds, and separately fails	§14
T15	Cross-tenant identity, key or evidence reference supplied	GI-9
T16	Provider model alias cannot be resolved; evidence records the weaker status	§21.1
T17	**Provenance forgery** — requester asserts P1 for a model-derived value with no resolver attestation; verifier derives P4 and denies at tier 4	§7.3
T18	**Tier forgery** — requester asserts tier 1 for an action the registry maps to tier 4; PDP and PEP derivations diverge and fail closed	§7.2
T19	**Alternative-path detection** — a change made through the target's native interface is detected by reconciliation within the declared interval	§15.4
T20	**Approval redirection** — a valid approval for one transaction is presented against a different transaction with identical parameters	§10.3
T21	**Broker suppression** — the PEP executes and then omits the execution receipt; the commit intent record and the completeness-monitor anomaly expose it	§20.2
T22	**Canonicalisation gadget** — a message with a duplicate member name that becomes valid after normalisation is rejected before signature verification	§12.6
T23	**Presentation mismatch** — the rendering shown differs from the canonical envelope; the presentation digest fails	§18.1
T24	**Cumulative budget** — n actions individually within `ceiling_per_action` exceed `ceiling_cumulative`; the reservation fails	§17.1
T25	**Segregation of duties** — the approver is the requester or the beneficiary at tier 5	§18.3
T26	**Business duplication** — the same supplier payment is issued twice under two different `transaction_id` values; the business deduplication key fails it closed	§13.1.1
T27	**Reference substitution** — a `value_ref` resolves to one plaintext at approval and another at commit; the commitment check fails	§9.4
T28	**Registry downgrade** — a decision is presented under a registry version revoked between decision and commit; the PEP fails closed	§7.5, item 4
T29	**Quarantined reservation** — an indeterminate transaction's budget reservation is not released on timeout, and a later target commit is reconciled against it without overspend	§17.1
T30	**Array reordering** — a semantically identical envelope with `parameters` in a different order produces the same digest, and a duplicate sort key is rejected	§12.4
T31	**Unattested delegation hop** — a chain in which one hop carries no workload attestation fails closed, rather than degrading to the nearest attested principal	§17.2

#	Test	Traces to
T32	**Approval metadata substitution** — `approver_role`, `auth_assurance_level`, `session_binding` or `approver_credential_ref` is altered after signing; the approval signature fails	§10.3
T33	**Adapter-mode downgrade** — `assurance_mode` is changed from `conditional-commit` to `broker-ledger` after approval; the approved-action digest fails	§10.2
T34	**Attestation substitution** — a different signed attestation asserting the same value and class is swapped in; the approved-action digest fails	§9.4, §10.2
T35	**Write-ahead bypass** — the PEP skips the Commit Intent Record entirely and attempts execution; without a valid execution proof the credential broker refuses, or, where the target cannot verify a proof, reconciliation surfaces the commit	§20.2.1
T36	**Unresolved commit intent** — a Commit Intent Record is written and no terminal receipt arrives within its window; a signed evidence anomaly is emitted although no sequence gap exists	§20.2.2
T37	**Concurrent business duplication** — two envelopes for the same business operation are submitted simultaneously; the atomic deduplication reservation admits exactly one	§13.1.1
T38	**Envelope time** — envelopes presented before `not_before`, after `expires_at`, with `issued_at` outside the permitted skew, and with `expires_at` preceding `issued_at` are each rejected with a distinct outcome	§13, step 5
T39	**Low-tier provenance omission** — a model-derived parameter at tier 2 with no attestation is recorded as `not-assessed` with no class, never as P2; it does not satisfy a requirement expressed as a class, and it is not rendered to an approver as though it had one	§7.4
T40	**Profile downgrade** — an envelope declaring a profile revision below the minimum accepted is rejected at step 3	§9.1.1, §22.3
T41	**Incomplete preconditions** — an envelope carrying one irrelevant precondition, while the registry declares three as required, fails closed	§7.5, §9.6
T42	**Attestation-value substitution** — a valid attestation for one value is presented alongside a different value in the parameter; the recomputed parameter-value commitment does not match and the envelope fails with `E-PROVENANCE` at step 9, before tier derivation	§7.3.1, §13 step 9
T43	**Decimal denormalisation** — `"1200.0"` and `"1200.00"` for a scale-2 currency produce the same canonical form and the same commitment; a value requiring truncation is rejected rather than rounded	§12.3, §7.3.1
T44	**Intent substitution** — after approval, the intent values are changed; `origin` is rewritten from `model-generated` to `user-confirmed`; the confirmation record is substituted; and the content behind `schema_uri` changes. Each independently invalidates the approval	§9.3, §10.2
T45	**Registry threshold matching** — two entries for the same action type and target, one `{always:true}` at tier 3 and one `amount ≥ 1000.00` at tier 4, resolve deterministically by most-specific match; a pair with no default entry fails closed with `E-TIER`; a unit or type mismatch fails closed rather than coercing	§7.5.1
T46	**Concurrent budget exhaustion** — two transactions, each individually within `ceiling_per_action`, together exceeding `ceiling_cumulative`, are submitted simultaneously; the atomic compare-and-reserve admits exactly one	§17.1
T47	**Reservation rollback** — a mandate rejection at step 8, an approval failure, an evidence-service outage and a precondition mismatch each occur after at least one reservation exists; both are released, the release is recorded, and a retry with the same business key succeeds. A crash between the two reservations converges on restart	§13.3
T48	**Execution-proof replay and misuse** — a proof is presented twice, to the wrong audience, after expiry, and for a different target; each is rejected with `E-PROOF` and the replay is recorded as a security event	§20.2.3
T51	**Signed denial** — a structurally valid envelope carrying a correctly signed, unexpired Policy Decision Record with `effect: "deny"` enters `Denied`, releases both reservations, and never obtains an execution proof. The same for `defer`, `indeterminate`, an unrecognised effect and an absent effect	§13.4, §13 step 11
T52	**Pre-authentication resolution** — an unauthenticated envelope names an external schema URI; no resolution of any kind occurs before signature verification at step 4, and the envelope is rejected at step 3 or 4 without a network request	§13 steps 2–6, §12.6
T53	**Undischarged blocking obligation** — a `permit` decision carrying a blocking obligation that is not discharged fails closed with `E-POLICY` before step 16	§13.4
T54	**Credential-scope widening** — a compromised enforcement point submits a Commit Intent Record naming a broader target scope; the issuer's independently derived `credential_scope_digest` differs and no proof is issued	§20.2.3
T55	**Credential reuse** — a credential released under one execution proof is presented for a second transaction and is refused	§20.2.3
T56	**Tier 5 quorum** — an automated tier 5 action carrying a single approval, or two approvals from the same credential or the same role, fails closed	§18.3
T57	**Currency mismatch** — an action whose `amount` carries `unit: "USD"` against a EUR mandate fails closed with `E-BUDGET`; and an action whose parameter `unit` disagrees with the named `unit_parameter` fails closed with `E-TIER`	§12.3, §17.1
T58	**Ambiguous registry** — two entries of equal specificity match the same action; the snapshot is rejected with `E-TIER` rather than resolved by name	§7.5.1(c)

#	Test	Traces to
T49	**Reconciliation interval bound** — a `broker-ledger` envelope declaring an interval above the policy maximum for its tier is rejected; and the interval appears in the rendering shown to the approver	§15.2, §18.1

Tests T17 to T25 are new in 0.2 and correspond to the defects found in 0.1; T26 to T31 are new in 0.2.1; T32 to T41 are new in 0.2.2 and correspond to the eleven normative gaps listed in §1.5; T42 to T49 are new in 0.3 and correspond to §1.6; T51 to T58 are new in 0.4 and correspond to §1.7. T26 is superseded for concurrency purposes by T37, and T24 by T46; both should be run as the sequential cases. Version 0.1's sixteen tests included "self-issued mandate" but no provenance-forgery test, no tier-forgery test and no approval-redirection test — which is to say, the three defects that most directly defeated its own control model were untested.

25. Conformance

No conformance scheme is defined in this edition. See §1.2.

An implementation MAY publish a **capability statement** of the form:

> *[Implementation, version] meets the requirements of GAIP Candidate Specification 0.4 sections [list], for action types [list], at impact tiers [list], with target adapters [list] in assurance modes [list], as assessed by [self / named party] on [date]. Profile revision accepted: [minimum]. Approval presentation position under §18.1.1: [(a) trusted rendering path — GI-5A only / (b) trusted approval client — GI-5A and GI-5H], per action type. State-freshness property held per adapter: [GI-6P / GI-6D with reconciliation interval and maximum detection latency]. Write-ahead evidence: [preventive, with a Credential Broker outside the enforcement point's trust domain / detective, by reconciliation]. Client and display integrity basis where GI-5H is claimed: [method]. Adapters operating in* `broker-ledger` *mode, with reconciliation intervals: [list]. Alternative paths declared under §15.4: [list]. Negative test results are available at [reference].*

The presentation position and the `broker-ledger` list are not optional detail. An implementation at position (a) cannot claim its approvals bind to what the approver saw, and an adapter in `broker-ledger` mode holds GI-6D and not GI-6P. A capability statement that omits either overstates what was assessed.

Such a statement is a description with a scope and a date. It is not a certification, and this profile provides no basis to describe it as one. "GAIP conformant", "GAIP compliant" or "GAIP certified" without that scope are not meaningful statements and should be treated as marketing.

26. Open questions and non-goals

26.1 Non-goals

GAIP does not prove that source data were truthful; that a model output was correct or unbiased; that a legal role or risk classification was correctly determined; that a human approval was wise or freely given; that separated authority roles cannot collude; that the target system faithfully implements its own receipt; or that an organisation is compliant.

It aims to make the authority and execution claim bounded, testable and comparable.

26.2 Open items

Load-bearing, in rough priority order:

1. **The normative schema and CDDL are not published** (§22.1). Until they are, this is a data model in prose, two implementations will diverge on structure and cardinality, and §13 step 2 can only be satisfied locally. This is the single reason the profile is **not yet implementable interoperably**.
2. **No transport binding** (§22.4). Interoperability requires at least one, including an asynchronous receipt path for `AwaitingReceipt`.
3. **Test vectors are partial** (§23) — nine vectors now cover canonicalisation, domain separation, the parameter-value commitment, the intent digest, presentation, key derivation, the registry and the execution proof, but there are still no signature, key-resolution, CBOR or end-to-end vectors, and no published test keys.
4. **No reference implementation, and none by a second party.** Vendor neutrality remains an objective, not a property (§1.2).
5. **Target receipt trustworthiness** (A9 in §4.2) is assumed and unaddressed. A target that misreports what it committed defeats the chain at the last hop, and no amount of upstream rigour repairs it.
6. **Evidence immutability against GDPR erasure and rectification.** The salted-commitment rule in §9.4 and the separation of payload from integrity evidence described in *Position Paper 01*, §3.5, are partial answers; this profile does not yet carry that separation as a normative requirement of §20.
7. **Cross-representation digest mapping** (§12.5) is prohibited rather than solved.
8. **Attenuation semantics** (§17.2) are stated as a monotonicity rule without a formal lattice. Two implementations may disagree on whether a given narrowing is a narrowing.
9. **Presentation profiles are per action type and unstandardised.** This is correct for domain fidelity and bad for interoperability, and the tension is unresolved. The trusted-approval-client position in §18.1.1 makes it worse, because the client must render identically to be useful.
10. **The Registry Authority is now the profile's highest-value single target** (§7.5, boundary 12 in Annex I). Its own governance — who may sign a snapshot, under what quorum, with what evidence — is stated as a requirement and not yet as a specification.
11. **Business deduplication keys** (§13.1.1) are declared per action type with no standard construction, so two implementations will deduplicate differently. 11a. **The evidence layer should probably not be ours.** RFC 9943 (SCITT) standardises signed statements, registration policies, transparency services, receipts and independent replay verification — the greater part of what §20 specifies by hand. A future edition should express the evidence layer as a SCITT profile rather than a parallel construction, and the authority semantics as the statement payload. That is a design decision this revision does not take. 11b. **The execution proof assumes a credential broker or target that can verify it** (§20.2.3). Against most third-party software-as-a-service no such verifier exists, and the control degrades to reconciliation. The profile says so; it does not solve it. 11c. **The registry's most-specific-match algorithm** (§7.5.1) counts scalar comparisons. That is deterministic but crude: two entries of equal specificity are resolved by `entry_id` ordering, which is arbitrary rather than meaningful. A future edition should either require non-overlapping conditions or define a genuine specificity lattice. 11d. `independence_domains` (§7.4) makes P0 checkable but pushes the judgement into registry configuration. Whether two named domains really are organisationally independent is not something this profile can verify.
12. **Conformance** (§25) — a scheme requires a test suite, an attestation format, an assessor role and product-versus-deployment separation. None exists.

26.3 The next release gate

This profile should not move beyond **Candidate** until all of the following exist:

1. a normative JSON Schema and CDDL;
2. a defined HTTP transport binding and an asynchronous receipt binding;
3. full JSON and CBOR signature vectors with published test keys;
4. the provenance-class derivation in §7.4 expressed as executable test cases, not only as a table;
5. an exact Action Type Registry format, signing profile and change-control specification;
6. a complete approval and presentation trust model, including a specified trusted approval client for position (b);
7. reconciliation semantics for quarantined budget reservations and indeterminate outcomes, specified rather than required;
8. a decision on whether the evidence layer becomes a SCITT (RFC 9943) profile rather than a parallel transparency construction;
9. at least one reference implementation, and one independent implementation attempt by a party that is not GoSec Cloud OÜ.

Items 1 and 2 are not editorial gaps. Without them, two correct implementations of this text can still produce structurally incompatible envelopes.

Criticism of any of the above, and of anything not on the list, is wanted at https://www.vianordis.eu/en/contact. Please cite the section number.

Annex I — Trust boundaries and compromise scenarios

I.1 Trust boundaries

#	Boundary	Scope
1	Human identity	Authentication and authorisation of natural persons and organisational roles
2	Agent identity	Registration, attestation and lifecycle of agents and instances
3	Mandate authority	Issuance, validation and revocation. Business applications must not assume this authority
4	Policy	Versioned decisions. Models must not alter or bypass policy
5	Execution	Action broker and tool gateways as the controlled path to targets
6	Key	Key management and signing outside business applications and ordinary service accounts
7	Evidence	Separate, append-only or comparably protected path with independent checkpoints
8	Tenant	Separation of identities, policies, keys, data, evidence and network paths
9	**Input trust**	**All content the agent reads — documents, mail, web pages, tool descriptions, retrieved records — is untrusted input. Anything derived from it is a claim, not a fact**
10	Target state	Impact-determining target state is revalidated at commit; approval-time state is not assumed to remain true
11	Time and cryptographic lifecycle	Clock sources, timestamp services, signing keys, validation material and renewal processes protected separately
12	**Action type registry**	**The signed mapping from action to tier, the declaration of impact-determining parameters, and the presentation profiles. Introduced in 0.2 and hardened in 0.2.1 (§7.5). This is the profile's highest-value single target: whoever controls it controls every tier-indexed rule, every impact-determining declaration and every rendering the approver sees**

I.2 Compromise scenarios and expected bounds

Compromised component	Expected bound
Single business application	Cannot issue mandates; cannot set its own tier or provenance class; no direct privileged target access
Single agent	No self-escalation; revocable; only mandated actions; chain attenuation holds
AI model or provider	No technical authority; bounded data context; substitutable
Service account	Short lifetime, bounded scope; no access to central signing keys
Enforcement point / broker	Cannot forge mandate, decision or approval (inner signatures preserved). Evidence suppression is detected by the Evidence Service completeness monitor (§20.2.2), and is **prevented** only where a credential broker or target verifies the execution proof (§20.2.1); against a target that cannot, the bound is detective, not preventive
Evidence producer	Independent chaining, signature or witness makes manipulation detectable
Tenant administrator	No cross-tenant rights; critical actions under quorum approval
Platform administrator	Key separation, four-eyes controls, independent evidence, constrained break-glass. **Bounded, not prevented**
Build or release pipeline	Signed releases, SBOM, separated approval, reproducible or verifiable builds
Input data and tool descriptions	**Not prevented.** The model may derive falsified parameters and a correctly bound approval will bind to them. Bounds: resolver attestation, out-of-band verification of impact-determining fields, provenance shown in the rendered approval, change thresholds on master data, tier ceilings for untrusted-input-derived actions
Target state change after approval	Approval becomes stale; broker revalidates; conditional commit where supported, `broker-ledger` reconciliation otherwise
Replay or duplicate delivery	Derived idempotency key and transaction uniqueness prevent a second commit
Clock manipulation	Monitored clock source, declared skew bound, signed checkpoint timestamps, independent timestamping at high assurance

Compromised component	Expected bound
Evidence signing-key compromise	Key separation, rotation, revocation, recorded compromise time, retained historical validation material, re-sealing
Break-glass credential misuse	Multi-party activation, short absolute expiry, real-time alerting, separate evidence, mandatory review

I.3 Limits of isolation

No zero-trust architecture removes all risk. Legitimate but overly broad data access can still be abused. An organisation with control of all administrative keys can destroy technical evidence unless independent checkpoints or external witnesses exist. And an agent acting with valid authority on false premises is outside the model entirely.

Annex II — Glossary

Action commitment — a digest of the operation and parameters presented to an approver.

Action tier — the impact class (0–5) of an action, derived from the signed action type registry, never asserted by the requester.

Actor chain — the ordered sequence of workload identities from the outermost actor to the executing one.

Break-glass — a declared, separately credentialled emergency path, with multi-party activation, short absolute expiry, alerting and mandatory review.

Derived class — the provenance assurance class a verifier computes from an attestation, as distinct from a class asserted by a requester.

Domain separation — the practice of prefixing a distinct label into a hash input so that a digest computed for one purpose cannot be presented as one computed for another.

Evidence Record — the correlated, signed record linking decision, approval, execution and outcome.

Governed Action Envelope — the signed, short-lived object carrying an action from decision and approval to the enforcement point.

Impact-determining parameter — a parameter whose value can change the impact tier, the policy outcome, the identity of the beneficiary, or the reversibility of the action.

Mandate — a bounded assignment linking an agent identity to a principal, with purpose, scope, budget, validity, delegation rules and revocation.

Precondition commitment — a digest or version reference for the material state that made an operation permissible.

Presentation digest — a digest over the artefact actually rendered to the approver, bound into the approval signature.

Principal — the natural person or organisational body on whose behalf an action is taken. Not a shared technical account.

Provenance assurance class (P0–P4) — how trustworthy the origin of a parameter value is; defined in *Position Paper 01*, §2.5.

Shadow Execution — an AI-assisted action that cannot be reconstructed from the organisation's records; defined in *Position Paper 01*, §2.2.

SVID — SPIFFE Verifiable Identity Document.

TOCTOU — time-of-check/time-of-use; the gap between validating a condition and acting on it.

Write-ahead evidence — the commit intent record written and checkpointed before any side effect.

Annex III — References

Normative for this profile

RFC 7493 (I-JSON) · RFC 7515 (JWS) · RFC 8785 (JCS, Informational) · RFC 9052 (COSE Structures and Process, STD 96) · RFC 9053 (COSE Initial Algorithms, Informational) · RFC 9338 (COSE Countersignatures, updates 9052) · RFC 9864 (Fully-Specified Algorithms, updates 9053) · RFC 3339 (date and time) · RFC 9110 §8.8.3, §13 (ETags and conditional requests) · JSON Schema 2020-12 · CDDL (RFC 8610) · ISO 4217 (currency codes) · ISO 8601 (durations)

Informative

RFC 9943 (SCITT, *An Architecture for Trustworthy and Transparent Digital Supply Chains*, Proposed Standard, IETF, 30 June 2026) · RFC 9396 (OAuth 2.0 Rich Authorization Requests, Standards Track, May 2023) · Commission Delegated Regulation (EU) 2018/389, Art. 5 (PSD2 dynamic linking) · Fatmi, A., *Faramesh: A Protocol-Agnostic Execution Control Plane for Autonomous Agent Systems*, arXiv 2601.17744 · Kaul, A., Lan, Q. and Gupta, P., *AgentBound: Verifiable Behavioral Governance for Autonomous AI Agents*, arXiv 2606.30970, June 2026 · RFC 2753 and RFC 3198 (policy framework terminology; origin of PDP/PEP) · ISO/IEC 10181-3:1996 (ADF/AEF) · OASIS XACML v3.0 (2013; v4.0 CSD 01, February 2026) · NIST SP 800-207 (August 2020) · RFC 8693 (OAuth 2.0 Token Exchange) · Birgisson et al. (2014), macaroons · SPIFFE/SPIRE · RFC 6962 (CT v1, Experimental) and RFC 9162 (CT v2, Experimental; formally obsoletes 6962, not yet the deployed profile) · RFC 3161 and RFC 5816 (time-stamping) · RFC 4998, RFC 6283 and RFC 9169 (evidence record syntax) · SLSA, in-toto, Sigstore · CycloneDX (Ecma-424), SPDX (ISO/IEC 5962:2021) · ISO/IEC 42001:2023 · ISO/IEC 27001:2022 (Amd 1:2024) · NIST AI RMF 1.0 (NIST AI 100-1, January 2023) · Model Context Protocol · Hardy, N. (1988), "The Confused Deputy", *ACM SIGOPS OSR* 22(4), 36–38 · ISO 20022 `pain.002`

Imprint

Published by **GoSec Cloud OÜ**, 9 Järvevana tee, 11314 Tallinn, Estonia. Estonian Commercial Register no. 17359627 · VAT EE102990704 · Registry court: Tartu District Court. Authorised representative: Yves Frédéric N'Soussoula. info@gosec.cloud

Author: **Yves Frédéric N'Soussoula**, Founder & CEO.

Cite as: N'Soussoula, Y. F. (2026). *Governed Action Interchange Profile — Candidate Specification 0.4*. Tallinn: GoSec Cloud OÜ. https://www.vianordis.eu

Review and criticism: https://www.vianordis.eu/en/contact. Please cite the section number. Corrections of fact are applied in the next edition and listed; substantive critique is credited unless the contributor prefers otherwise.

End of candidate specification — version 0.4