MURAQIB Security Whitepaper

Architecture, Threat Model, and Independent Verification

MURAQIB — the clearing layer for autonomous agent action, operated by OQIRON.

D2 · Security whitepaper · content of 2026-09-18 · source 8132aa73

> **How to read this document.** Every capability claim in this paper carries one of three markers, and no claim appears without one. **As built** — what runs in production at the date on this page. These claims are written to survive a reader with access to the source, the database, and the running service. Where something is partial, dormant, or absent, it is named as such rather than omitted. **Objective** — what the clearing layer is designed to become. Written in the future tense, and describing capability that does not exist yet. **Invariant** — design constraints, exclusions, and contracts intended to hold at every stage of the roadmap. A section may contain claims in more than one register; where it does, each is marked separately. Section 11 collects every limitation stated anywhere in this document into a single place, so that a reviewer who reads nothing else can read that. We would rather be read that way.

Executive Summary

MURAQIB is a pre-execution decision point for AI agent actions. An agent submits a proposed action — a payment, a message, a published document — and receives a verdict before it acts: approved, blocked, or escalated to human authorisation. Every verdict is sealed into an append-only evidence chain, and a recipient who retains verification material independently can confirm that chain against a signing key present on no OQIRON production host.

What MURAQIB is not, stated before anything else: it is not an enforcement point. It issues decisions; it does not execute or block. There is no proxy in the network path, no egress filter, no credential broker. An agent that receives a refusal and acts anyway is not stopped, and an agent that never asks is not observed. Enforcement rests on the calling system honouring the verdict — a contract the supplied SDK is built to make difficult to violate accidentally, since every failure mode in the client library, including an unreachable rail, resolves to a non-verdict the caller must treat as "do not proceed" — but which a determined caller bypasses simply by not calling.

What the design gives instead is evidentiary rather than preventive. The verdict is issued before the action and sealed whether or not it is honoured, so a caller that proceeds against a refusal faces a sealed record of the refusal it received, timestamped and outside its own control, countersigned at the next anchoring ceremony. That record does not by itself establish that the action was performed; correlating effects against clearances is the institution's work, and some actions leave no correlatable effect. What it removes is the possibility of a governed action being later described as unremarked.

This is the same structure that makes financial messaging infrastructure work. A clearing rail does not physically prevent a bad transaction. It makes a submitted one impossible to perform unrecorded, and that turns out to be enough to change institutional behaviour.

Three properties distinguish MURAQIB from post-hoc audit:

The decision precedes the action. Audit tells an institution what went wrong. A pre-execution decision point gives it the opportunity to have gone right, and a record of the choice either way.

A decision that cannot be sealed is not a decision. If the evidence write fails, the request fails and nothing proceeds unsealed. One exception is disclosed: a failure to resolve tenant identity records a sentinel and permits evaluation to continue, which is defensible only while no control reads tenant identity. Section 11 records it.

Verification does not require trusting the operator. The evidence chain's format is fully specified in Appendix B, the verification tool is published under an open licence, and the signing key is held offline by a custodian. A third party holding this document and the published key can recompute the chain and check its signatures without OQIRON's cooperation or consent — provided it retained verification material of its own beforehand. A recipient who retains nothing is trusting OQIRON.

This document states what is built and what is not, in every section, using three registers: **As built**, **Objective**, and **Invariant**. No claim appears unmarked. Section 11 collects every limitation stated anywhere in the document into a single register, including the ones that would give a careful reader pause. The most significant are named there first, not buried.

The roadmap follows from that register. The next engineering phase addresses: verifiable policy execution, closing the gap between tamper-evident logging and proof that the recorded policy is the policy that ran; a canonical action-intent digest binding the cleared action to its parameters, which closes parameter binding, replay and consumable clearance together; completion of tenant read isolation, where the machinery is proven and six routes are wired with twelve remaining; and rotation of credentials present in historical backups. Two objectives follow rather than accompany that work. Cryptographic binding of authorising identity — so that four-eyes distinctness is enforced over identities the rail can verify rather than identities a client asserts — is recorded in this document as not started. An enforcement layer, the network position that would make a refusal physically binding, is a deliberate later step, and one that depends on the evidence layer being correct first.

Readers evaluating MURAQIB for production use should begin with Section 11.

الملخص التنفيذي

MURAQIB هو نقطة قرار تسبق التنفيذ لأفعال الوكلاء الذكيين. يقدّم الوكيل الفعل المقترح — دفعة مالية، أو رسالة، أو مستند منشور — فيتلقّى حكماً قبل أن يُقدم على الفعل: approved أو blocked أو escalated إلى تفويض بشري. يُختم كل حكم في evidence chain لا تقبل إلا الإضافة، ويستطيع المتلقي الذي يحتفظ بمادة تحقّق مستقلة أن يتثبّت من هذه السلسلة مقابل مفتاح توقيع لا يوجد على أي خادم إنتاج تابع لـ OQIRON.

وما ليس عليه MURAQIB، ويُذكر قبل أي شيء آخر: إنه ليس نقطة إنفاذ. فهو يصدر القرارات ولا ينفّذ الفعل ولا يمنعه. لا يوجد proxy في مسار الشبكة، ولا مرشّح للصادر، ولا وسيط لبيانات الاعتماد. والوكيل الذي يتلقّى رفضاً ثم يمضي في الفعل لا يُوقَف، والوكيل الذي لا يسأل أصلاً لا يُرصَد. يقوم الإنفاذ اليوم على التزام النظام المستدعي بالحكم — وهو التزام صُمّم الـ SDK ليجعل خرقه غير المقصود صعباً، إذ إن كل حالات الإخفاق في المكتبة، بما فيها تعذّر الوصول إلى الـ rail، تُفضي إلى لا-حكم يجب على المستدعي أن يعامله على أنه "لا تمضِ" — غير أن المستدعي المصمّم على التجاوز يتجاوزه بألا يسأل ابتداءً.

وما يقدّمه التصميم بدلاً من ذلك إثباتيٌّ لا مانع. فالحكم يصدر قبل الفعل ويُختم سواء أُخذ به أم لم يُؤخذ، فالمستدعي الذي يمضي رغم الرفض يواجه سجلاً مختوماً للرفض الذي تلقّاه، موسوماً بوقته وخارجاً عن سيطرته، ومُوقَّعاً بالتوقيع المضاد في مراسم الترسية التالية. وهذا السجل لا يثبت بذاته أن الفعل قد وقع؛ فمطابقة الآثار بالأحكام عملٌ تقوم به المؤسسة، وبعض الأفعال لا تترك أثراً قابلاً للمطابقة. لكن ما يزيله هو إمكانية وصف فعل خاضع للحوكمة لاحقاً بأنه مرّ دون تسجيل.

وهذه هي البنية نفسها التي تجعل بنية المراسلات المالية تعمل. فالـ clearing rail لا يمنع المعاملة السيئة مادياً، لكنه يجعل المعاملة المقدَّمة مستحيلة الوقوع دون تسجيل — وقد تبيّن أن هذا القدر كافٍ لتغيير سلوك المؤسسات.

ثلاث خصائص تميّز MURAQIB عن التدقيق اللاحق:

القرار يسبق الفعل. التدقيق يخبر المؤسسة بما وقع من خطأ. أما نقطة القرار السابقة للتنفيذ فتمنحها الفرصة لأن تكون قد أصابت، وسجلاً للاختيار في الحالتين.

القرار الذي يتعذّر ختمه ليس قراراً. فإن أخفقت كتابة الدليل أخفق الطلب، ولا يمضي شيء دون ختم. ويُفصح عن استثناء واحد: إخفاق تحديد هوية الـ tenant يسجّل قيمة دالة ويسمح للتقييم بالمضي، وهو أمر يظل مقبولاً ما دام لا يقرأ أي control هوية الـ tenant. القسم الحادي عشر يسجّل ذلك.

التحقّق لا يستلزم الثقة بالمشغّل. صيغة الـ evidence chain موصّفة بالكامل في الملحق ب، وأداة التحقّق منشورة برخصة مفتوحة، ومفتاح التوقيع محفوظ خارج الاتصال لدى أمين عهدة. ويستطيع طرف ثالث يملك هذه الوثيقة والمفتاح المنشور أن يعيد حساب السلسلة ويتحقّق من توقيعاتها دون تعاون OQIRON أو إذنها — شريطة أن يكون قد احتفظ بمادة تحقّق خاصة به مسبقاً. والمتلقي الذي لا يحتفظ بشيء إنما يثق بـ OQIRON.

تذكر هذه الوثيقة ما هو مبنيّ وما هو غير مبنيّ، في كل قسم منها، عبر ثلاثة سجلات: كما هو مبنيّ، وهدف، وثابت. ولا يرد فيها ادعاء بغير وسم. ويجمع القسم الحادي عشر كل قيد مذكور في أي موضع من هذه الوثيقة في سجل واحد، بما في ذلك القيود التي تستوقف القارئ المدقّق. وأشدّها أثراً مذكورة في صدره، لا مدفونة في آخره.

وتنبثق خارطة الطريق من ذلك السجل. تتناول المرحلة الهندسية التالية: التنفيذ القابل للتحقّق للسياسة، وهو ما يسدّ الفجوة بين التسجيل الذي يكشف العبث وبين إثبات أن السياسة المسجّلة هي السياسة التي عملت فعلاً؛ وcanonical action-intent digest يربط الفعل المخلَّص بمعاملاته، فيغلق ربط المعاملات وإعادة التشغيل والتخليص القابل للاستهلاك معاً؛ واستكمال عزل القراءة لـ tenant، حيث ثبتت الآلية ووُصلت خمسة مسارات وبقيت ثلاثة عشر؛ وتدوير بيانات الاعتماد الموجودة في النسخ الاحتياطية التاريخية. أما الربط التشفيري لهوية المفوِّض، وطبقة الإنفاذ — أي الموضع الشبكي الذي يجعل الرفض ملزماً مادياً — فهدفان لاحقان، والثاني منهما يتوقف على صحة طبقة الأدلة أولاً.

وعلى من يقيّم MURAQIB للاستخدام الإنتاجي أن يبدأ بالقسم الحادي عشر.

---

1. What MURAQIB is for

1.1 The problem is not that agents make mistakes

Autonomous software agents now draft, decide, and act. They move money, send communications on an institution's behalf, query and export data, and modify records in systems of record. The capability is real and it is arriving faster than the controls around it.

The instinctive response is to make the agents themselves safer — better models, better prompts, better guardrails inside the agent. That work matters and it is not sufficient. It fails for a structural reason: **the control regime around an agent is chosen, configured, and controlled by the party that benefits from the agent acting.** Model vendors supply constraints; the deploying organisation decides which to apply, how to configure them, and whether the next agent it deploys uses them at all.

Institutions are not without controls. Identity systems, API gateways, service meshes, workflow approvals, secrets management, egress filtering and log aggregation are all mature, and a competent enterprise already centralises several of them. The gap is narrower and more specific than "nothing exists":

There is no widely adopted, vendor-neutral clearance contract that applies the same action-level governance and the same evidence semantics across heterogeneous autonomous agents. An institution running forty agents from six vendors can centralise identity and network egress, and still have no single place to ask what it permitted at the level of the action itself — no common vocabulary for "this agent proposed to move this amount to this counterparty," no uniform record of what was decided, and nothing an outside party can verify without trusting the institution's own systems.

The second instinctive response is to log everything and audit afterwards. This is the default posture for software systems, and it does not transfer to consequential agent action. **After-the-fact audit assumes the action is reversible, or that the delay before discovery is tolerable.** For a wire transfer that has settled, a customer list that has left the building, or a production table that has been dropped, neither assumption holds. An audit trail that tells you precisely how you lost the money is not a control.

Institutions already know this. The banking sector's own answer to consequential human action is not the audit trail but the maker-checker: a second person, before execution, on the actions that matter. Pre-execution control is the established answer for humans. It has not yet been built as shared infrastructure for agents.

1.2 The clearing layer

Objective. MURAQIB will be a clearing layer for autonomous agent action: an independent rail through which every consequential action passes before it executes.

The analogy to financial market infrastructure is institutional and protocol-level, not a claim of functional equivalence. What transfers is the shape: standardised instructions, registered participant identity, common rules applied by infrastructure the counterparties do not control, and a durable record none of them can quietly amend. What does not transfer is the division of labour — in payments, messaging, authorisation, clearing, settlement and supervision are distributed across several distinct institutions and systems. MURAQIB combines the decision and evidence functions in one rail, and the analogy should be read for the institutional form rather than the mechanics.

The property being borrowed is this: infrastructure of that kind does not make a participant trustworthy. It makes a participant's behaviour *checkable* — and that is what allows institutions who do not know each other to transact at scale.

Invariant. Four properties follow. They are design constraints on any workable solution, not features of a particular release:

R1 — Pre-execution. The decision point sits before the action, not after it. A verdict that arrives after settlement is a report, not a control.

R2 — Independence of the action channel from the policy channel. The rail is not a library inside the agent. It is a separate system with its own registered identity for each agent, its own evaluation, and its own record. **The channel through which an agent requests clearance cannot alter the policy used to evaluate that request, cannot grant itself clearance, and cannot write to the record.** Administrative control over the catalogue is a distinct boundary, governed separately; see §1.3.

R3 — Uniformity of contract. One clearance contract, identical for every agent regardless of who built it, what framework it uses, or whether the rail runs hosted or inside the institution's own perimeter. R3 requires *protocol* equivalence. It does not assert that operational trust boundaries are identical across deployments: custody of keys, administrator access, failure domains and regulatory boundary differ between hosted and in-perimeter deployment, and are stated separately in the deployment section.

R4 — Independently verifiable evidence. Every request and every verdict is sealed into a chain record whose integrity a third party can confirm using verification material obtained independently of the party presenting the record. This holds for decisions taken since sealing began; the unsealed operational store from which the interface reads also retains earlier records that carry no seal, and the verifier reports their count rather than including them silently (§11.2.20).

R4 requires a precise statement, because a looser reading of it does not survive §3.4. A hash chain establishes integrity **relative to a countersigned checkpoint**. It does not establish that the checkpoint presented is the most recent one. Operator-independent verification therefore requires two things from the verifier, neither of which the presented file can supply: the public key, obtained from a channel independent of the bundle; and knowledge of the latest checkpoint's date and coverage, likewise obtained independently — or a checkpoint the verifier retained before the period in dispute. **Absent that, a recipient establishes that the records shown are authentic and unaltered, not that they are all of them.** §3.4 states the consequence for the operator case and §9 states it for the tool.

1.3 Policy authorship, policy integrity, and policy authorisation

Invariant. R2 invites an obvious objection, and it deserves answering before a reader raises it.

If the control catalogue is authored by the deploying institution — and it must be, since only that institution knows its risk appetite, its regulatory obligations and its thresholds — then the policy governing the agents is written by the same party that benefits from the agents acting. The indictment in §1.1 appears to land on the rail itself.

Three distinct questions are tangled here, and separating them is the whole answer.

Authorship — who writes the policy. The institution, necessarily. R2 makes no claim otherwise.

Integrity — whether the policy that was applied is the policy on record. **As built**, this is established by two mechanisms. A signed pin binds the deployed catalogue to a digest countersigned with a key held offline: a verification tool recomputes the digest from the deployed files and fails if it differs, and this has been demonstrated by inducing the failure rather than only observing the pass — a single altered verdict inside one control's decision function is detected and the control is named. And since 8 August 2026, every change to the deployed catalogue is itself sealed into the evidence chain, recording the digest before and after, whether the change carried a valid signature, and — where the ceremony was followed — who made it and why. The first such record is sealed at chain sequence 457.

Authorisation — whether a change to the catalogue was approved by anyone entitled to approve it. **As built, this does not exist.** There is no maker-checker, no role separation, and no approval workflow governing changes to the catalogue. The rail applies structural four-eyes requirements to agent actions and applies none to its own rules.

The consequence must be stated rather than left for a reader to derive. **An administrator who can change the catalogue can loosen a control, have an agent perform an action that is then legitimately cleared under the policy in force, and restore the catalogue afterwards.** The evidence chain remains cryptographically sound throughout: it faithfully records that a permissive policy was applied. Since 8 August 2026 it also records both catalogue changes, so the sequence appears in the record as two policy events bracketing a decision — but this is *detection by an examiner who looks*, not prevention, and §3.2 states the further limit that the same administrator can modify the mechanism that records it.

Objective. Enforced role separation and dual authorisation for catalogue changes, so that a single administrator cannot unilaterally alter the policy in force. Not started. §11 records this as an open limitation.

1.4 The institution layer

Objective. A rail is a technical artifact. What will make shared infrastructure credible is the institution that operates it: a body that publishes the clearance standard, defines what it means for an agent to be admitted to the rail, certifies conformance, and holds the evidence in a form regulators and counterparties can independently verify. The network without such an institution would be middleware; the institution without the network would be a standards committee with no leverage. OQIRON is being built to occupy that position, and the MURAQIB Integration Standard — one contract, identical for agents OQIRON builds and agents third parties build — will be its technical expression.

As built. One element of that posture exists today and can be checked without asking OQIRON's permission. The verification tool described in Section 9 is published under the MIT licence at `github.com/oqiron/muraqib-verify`, and the anchor public key is published at `oqiron.ai/verify` — a channel deliberately separate from any evidence file OQIRON sends, so that a recipient can confirm the key without relying on the file it validates. An institution whose records can only be checked with its own permission is asking for trust rather than earning it.

Objective — and named here because a reader should not have to ask. The governance of OQIRON itself — ownership, neutrality commitments, the process by which the clearance standard is amended, participant representation, independence of certification, and resistance to capture — is a question of institutional legitimacy rather than system security. §3.4 establishes that it is also a security dependency: an operator holding the sole signing key is constrained by external retention of checkpoints, not by cryptography. A governance framework addressing this is planned and **not yet published**. It is named here so that a reader evaluating the institutional claim knows the question is recognised and unanswered in this paper, rather than overlooked.

1.5 Who runs on the rail

Objective. The rail is intended to serve three constituencies. The order below is a hardening sequence, not a commercial queue: each stage subjects the rail to a class of scrutiny the next depends on.

Regulated institutions first — banks, insurers, healthcare providers, government entities. They present the sharpest problem: consequential actions, an existing four-eyes culture, and a supervisor who will ask how autonomous systems are controlled. They also apply the most demanding evidentiary

standard, because "we governed it" must be demonstrable to a third party. Infrastructure that survives that scrutiny can be trusted more broadly.

Enterprises generally — any organisation whose agents touch money, personal data, or customers. The requirement is the same; the forcing function is commercial and reputational rather than supervisory. This stage establishes that the integration contract holds across a wide diversity of agent architectures.

Sovereign and national AI infrastructure — the destination, and the reason the two prior stages exist. Where a country deploys agentic systems across government and critical sectors, the question of who cleared what, under which policy, on whose authority becomes a matter of national accountability. It is also the deployment least tolerant of an unproven rail, which is why it follows rather than leads.

A sovereign or national deployment raises questions this section does not answer: where evidence resides, who holds the keys that protect it, whether the rail's operator can read a tenant's evidence, what an on-premise deployment requires in outbound connectivity, how tenant separation differs from physical isolation, and what becomes of the evidence on termination. Those are addressed in §10; they are named here because they determine whether an architecture is sovereign at all, and because §10 discloses that tenant read isolation is presently not enforced.

1.6 What MURAQIB is not

Invariant. Three exclusions hold at every stage of the roadmap.

MURAQIB **does not make an agent correct.** It governs what an agent is permitted to do, not whether the agent's reasoning was sound. An agent that correctly determines it should transfer funds and one that reaches the same conclusion by hallucination present identically at the rail.

MURAQIB **does not establish that a recorded fact was true.** The evidence chain establishes that a request and a verdict were recorded in a particular order and, relative to a countersigned checkpoint, have not been altered since. It does not establish that the request accurately described the action the agent went on to perform. Resource-side enforcement (§2.1) narrows this materially for governed resources — where a resource releases only against a clearance bound to specific parameters, the declared action and the executed action converge — but the residue never disappears: the real-world meaning of an action, beyond what a governed interface can check, is outside what any cryptographic mechanism can establish.

MURAQIB **is not a blockchain and does not require one.** The evidence record is a hash-linked append-only sequence, countersigned periodically with a key held offline. It requires no distributed consensus, no token, and no third-party network. Two consequences follow directly and are stated here rather than buried. **A hash chain establishes integrity relative to a trusted checkpoint** — records covered by a countersigned checkpoint cannot be altered without detection, while records added since the most recent independently retained checkpoint sit within a tamper window whose width is the countersignature interval. And **a chain cannot establish that it has been shown in full**: a record truncated at an authentic earlier checkpoint is, from the inside, indistinguishable from a complete one. Section 7 gives the interval and the construction; §9 states how the verification tool reports both limits.

---

2. The trajectory: decision today, enforcement tomorrow

This section states the single most important thing a technical reader needs before evaluating anything else in this paper, because it determines what every other claim means.

2.1 Decision point and enforcement point

Established access-control architectures distinguish two roles. A **Policy Decision Point** evaluates a request against policy and returns a verdict. A **Policy Enforcement Point** sits in the path of the action and prevents it proceeding without one. The separation is deliberate: decision logic is centralised, auditable and changed in one place, while enforcement is distributed to wherever the resource lives.

As built, MURAQIB is a Policy Decision Point. It authenticates the calling agent, evaluates the declared action against its control catalogue, returns a verdict of APPROVED, BLOCKED, or ESCALATED, and seals the request and the verdict into the evidence chain. It does not intercept the agent's outbound call. There is no proxy in the network path, no egress filter, and no credential broker.

The consequence, stated plainly: **an agent that receives BLOCKED and performs the action regardless is not stopped by MURAQIB.** Enforcement today rests on the calling system honouring the verdict — a contract the SDK is built to make difficult to violate accidentally, since every failure mode including an unreachable rail resolves to a non-verdict the caller must treat as "do not proceed" — but which a determined caller bypasses simply by not calling. **What the decision point does give is evidentiary rather than preventive**: the verdict is issued before the action and sealed whether or not it is honoured, so a caller that proceeds against a refusal has a sealed record of the refusal it received — timestamped, outside its control, and countersigned at the next anchoring ceremony. That record does not by itself establish that the action was performed — correlating effects against clearances is the institution's work, and some actions leave no correlatable effect — but it removes the possibility of a governed action being later described as unremarked. §11 records the absence of an enforcement point as the first item a reader assessing production use should resolve.

As built — an operational consequence of that fail-closed contract. Because an unreachable rail resolves to "do not proceed," rail unavailability becomes unavailability of every governed action. This is correct security behaviour and it makes the rail availability-critical infrastructure: an adversary who cannot defeat authorisation may still attack availability. §10 states the deployment topology, capacity bounds and recovery posture as built; §11 records the dependency as a limitation rather than leaving a reader to discover it.

Objective. Full resource-side enforcement is the defining capability of the finished system, and the one that most distinguishes it from what runs today. In the target state the consequential action will not execute without a valid clearance.

Three things about that objective, stated so a reader can judge feasibility rather than infer it:

Status: not started. No component of resource-side enforcement exists in the codebase as of this draft. There is no clearance receipt, no token issued on approval, no credential broker, no proxy, and no signed per-decision artifact — not implemented, not stubbed, not scaffolded. An APPROVED verdict today is a JSON response and a sealed evidence record; it confers nothing a downstream resource could validate.

Mechanism: mediated enforcement, not resource-native adoption. Two architectures could satisfy the objective. In the first, the resource itself validates MURAQIB clearance natively — which would make the roadmap a matter of ecosystem adoption by payment networks and SaaS vendors, measured in years. **The design assumes the second:** the institution places OQIRON enforcement components — a credential broker, an egress control point — between its agents and its own resources, which it can do without any third party's cooperation. The precondition, and it is a real one, is that the mediated path must be the *exclusive* path: enforcement prevents bypass only where alternative credentials, alternative network routes and direct resource interfaces have actually been eliminated. That is a deployment obligation on the institution, not a property the rail can confer. A future revision of this paper will need to threat-model the enforcement components themselves, since they become part of the authorisation boundary.

Binding: to the action's parameters, not merely to the existence of a clearance. A clearance will bind the specific parameters of the action — amount, beneficiary, recipient — with audience, expiry, single-use semantics and replay resistance, so that a valid clearance for one transfer cannot release a different one, cannot be replayed, and cannot outlive its window. The clearance artifact schema is not yet designed. The deployed system's assertion signing demonstrates that structured bytes can be signed and bound to an action class, which is a necessary but not a sufficient condition for the above; the difficult parts — parameter canonicalisation, single-use semantics, and the window between clearance and execution — remain to be designed.

2.2 Why the decision-point stage is nevertheless load-bearing

As built. Two properties hold now, without resource-side enforcement.

Actions submitted are governed and sealed. For every action a caller submits, the verdict is evaluated against a stated catalogue and both request and verdict are written into a tamper-evident record. As of the correction described in §6, a verdict is never returned unless it was sealed: the evidence write and the chain seal commit in a single transaction, so a caller cannot receive a verdict the record does not contain.

Some bypasses are detectable after the fact; some leave no trace at all. The evidence chain is a complete record of what passed through the rail. Where an institution holds complete effect-side records and can reliably correlate them with clearances — a payment in the ledger, a change in the database, a message in the mail system's logs — a bypass may be identified as an effect with no corresponding clearance. That correlation is not automatic: it requires common identifiers, comparable parameters, consistent ordering, and effect-side records that are themselves trustworthy.

And it does not hold uniformly. Data egress — a file copied out, a customer list exported through an ungoverned path — frequently produces no reconcilable footprint at all. Nothing was paid, nothing changed in a system of record, and the loss is invisible to reconciliation. It is also, as §1.1 notes, among the least reversible actions an agent can take. **Data egress is therefore the weakest case for detection and the strongest standing argument for resource-side enforcement**, and it is a principal driver of the enforcement roadmap rather than an inconvenience to it.

Two further honesties. This is *detection*, not prevention — it establishes that something happened outside the rail, after it happened. And **the correlation is presently the institution's own work**: OQIRON does not today ship a reconciliation surface that ingests effect-side records and compares them against the cleared record. The chain makes reconciliation possible; it does not perform it.

2.3 The integration contract across the transition

As built. Ignoring the clearance contract is not prevented today. A caller that receives BLOCKED and proceeds, or that never calls at all, is not stopped. Where the resulting effect is independently observable and correlatable, a bypass may later be inferred by reconciliation; where it is not — see the data-egress case above — no record of the bypass exists anywhere, because the rail was never asked.

Objective. Under resource-side enforcement the same behaviour produces no action rather than an uncleared one.

Objective — the mechanism, since the compatibility claim is not credible without it. Enforcement requires the resource, or its mediator, to be presented with evidence of clearance, or the caller to obtain scoped short-lived credentials as the product of clearance. That boundary is designed to sit **inside the SDK**: the client will attach the clearance artifact or exchange it for credentials, so that caller code above the SDK is unchanged. **This is a compatibility objective, not an as-built guarantee** — the artifact does not exist yet, and a caller integrating against raw HTTP rather than through the SDK will not receive the abstraction. The Integration Standard's versioning and deprecation rules are stated in the integration section.

Invariant. R3 constrains every future change: the contract is identical hosted or on-premise, and identical for agents OQIRON builds and agents third parties build. As stated in R3, this is protocol equivalence and not equivalence of operational trust boundaries.

2.4 The trust ladder

Four-eyes authorisation — a distinct human countersigning a held action — is the central governance control for consequential actions. How much the rail can rely on it depends entirely on how the authorising human's identity reaches the rail, and the ladder below is a statement of that provenance and nothing more.

Invariant — Rung 1 semantics. The authorisation is asserted in the request payload; the authoriser's identity is attested by the calling system and not otherwise established.

As built at Rung 1. The rail evaluates the structural requirements: that an authoriser is named, holds a role permitted for the action, and is not among the named makers. A request failing any of these is refused.

Invariant — Rung 2 semantics. The assertion is cryptographically bound to the *calling agent*, so that its origin and integrity are established.

As built at Rung 2. The agent holds an Ed25519 private key registered with the rail and signs a canonical serialisation of the authorisation assertion which includes the action type; the rail verifies that signature against the registered public key before policy evaluation, and refuses an unsigned or invalid assertion rather than downgrading it. A signature produced for one action class does not verify when

presented for another — demonstrated, not asserted. §8 specifies the canonical byte construction, the signed field set, key registration and rotation, and states what the current construction does and does not establish about replay within a single action class.

The boundary, and it is the claim in this paper most easily over-read. Rung 2 establishes that *the registered agent asserted that a named human authorised this action*. It does not establish that the human did. The rail has no channel to that human, holds no credential of theirs, and cannot distinguish a genuine authorisation from one a compromised agent fabricated and signed with its own key. **At Rungs 1 and 2 the rail enforces the structural requirements of a four-eyes assertion; it does not independently establish human authorisation.** That property arrives only at Rung 3, and "verified human four-eyes authorisation" should not be read into anything below it.

As built — coverage. Rung 2 is currently enforced for **five action classes**: authorised external correspondence, document LLM egress, memory ingest, screening disposition, and campaign recipient authorisation. The fifth was brought under the requirement on 8 August 2026; §11 records the two approvals that predate that correction. All other action types operate at Rung 1: the authoriser fields are evaluated as supplied, and a signature, where one is present, does not alter the verdict. Six of eight production identities hold a registered key; the remaining two cannot use the Rung 2 classes at all.

There is presently **no configuration mechanism to require a minimum assurance rung for a given action class.** Coverage is determined by the catalogue itself and changes only by changing the catalogue — which, per §1.3, is not subject to approval or role separation. The two limitations compound: an administrator able to edit the catalogue can lower the assurance required for an action class. §11 records them together.

Objective — Rung 3. The authorising human authenticates directly to the rail through federated identity, so that the authorisation binds to a credential the human holds rather than to an assertion an agent makes about them. Not started.

Invariant. Once authoriser provenance has satisfied the rung required for the action class, the downstream policy evaluation is identical at every rung. Reaching a higher rung changes how provenance is established, not how the action is judged. Rung 3 will introduce a human authentication flow that does not exist today, so it is a compatibility objective rather than a guarantee that no integration work is required.

2.5 What this means for the rest of this document

Sections 3 through 10 are predominantly in the **as built** register, and each claim is marked. Where the target state differs materially from the current one, the difference is stated in the section rather than deferred.

A reader evaluating MURAQIB for deployment should assess it as what it is today: **an independent pre-execution decision point with a verifiable evidence record, whose enforcement depends on caller cooperation.** A reader evaluating OQIRON as a partner should assess the trajectory in §2.1 and the honesty of the accounting in §11; we would rather be judged on the latter.

3. Threat model

As built throughout, except where a mitigation is explicitly marked **Objective**. This section states what the system defends against, what it detects but does not prevent, and what it does not address. A reader who finds an adversary here that we have not considered should tell us; a reader who finds one we have understated has found a defect.

3.0 How to read this section

Each adversary is stated as a capability, not a person. "Database superuser" means anyone in possession of that capability, whether an employee, a contractor, an attacker who has escalated to it, or the operator of the rail. We do not distinguish by intent, because the system cannot.

For each adversary we state four things: what they can do, what the design **prevents**, what it **detects** without preventing, and what it **does not address**. The third and fourth are where the useful information is.

Two properties recur, and it is worth stating them once.

Isolation of the anchor private key is the principal cryptographic trust boundary protecting already-anchored history from a compromise of the production environment. The private half of the countersigning key is held outside the production environment and has never been present on any OQIRON server; signing is a manual act performed by a human on a separate machine. That isolation is necessary but not sufficient: the property also depends on correct checkpoint construction, correct canonicalisation, authenticity of the public key as the verifier obtained it, correctness of the verifier itself, and the integrity of the signing workstation and ceremony. §3.4 and §3.7 address the last of these directly.

The rail is a decision point, not an enforcement point. As stated in §2.1, MURAQIB does not sit in the path of the action. No adversary below is prevented from *performing* an action; they are prevented from obtaining a clearance for it, or from altering the record of what was decided.

3.1 A compromised or malicious agent

Capability. The agent holds a valid API credential and, in six of the eight production identities, a registered signing key. It composes its own requests and controls everything in the request body. **A credential's recorded scopes do not narrow this capability.** The scope test at the clearance routes applies only to a credential presented as an API key; exchanging the same credential for a bearer token, which any holder may do in one call, reaches those routes with no scope test at all. A reader must not treat a narrowly scoped credential as a narrowly capable one. See 11.2.23.

As of 13 September 2026, seven of those eight identities hold no credential at all: they remain registered and signature-capable but cannot transact until one is issued. One production credential is in force. This section describes the capability of an agent that holds one.

Prevented.

Self-granted clearance. The verdict is computed by the rail from the policy catalogue. An agent cannot supply a verdict, cannot suppress a control, and cannot see the evaluation before it receives the result.

Identity substitution. The agent identity used for evaluation and sealed into the record is taken from the authenticated credential, not from any field in the request body. An agent that names a different agent in its payload is evaluated as itself; the mismatch is computed and recorded. The body-supplied field is retained for diagnostic comparison, not consulted for identity.

Tenant substitution. Tenant attribution is derived server-side from the authenticated agent. A client-supplied tenant identifier is ignored and the attempt is logged.

Unsigned authorisation on the five Rung 2 classes. For authorised external correspondence, document LLM egress, memory ingest, screening disposition and campaign recipient authorisation, an assertion that is unsigned, invalid, stale beyond the permitted clock skew, or signed by an unregistered key is refused — refused, not downgraded to a lower assurance path.

Replay of an assertion across action classes. The signed canonical form includes the action type, so a signature produced for one action class does not verify when presented for another. This was demonstrated by substitution rather than assumed.

Escalation as a resumption path. An ESCALATED verdict is terminal at the API. There is no endpoint that later converts it to an approval, no callback and no polling path. An agent cannot wait out an escalation. §5 states what an institution does with an escalated action, since it is a governance state rather than a protocol state.

Detected but not prevented.

Misdescription of the action. The agent declares what it intends to do; the rail evaluates the declaration. An agent that declares a low-risk action and performs a different one receives a clearance for the declared action, and the record contains the declaration, not the act. Detection depends on the institution correlating effects against clearances (§2.2), and for actions with no reconcilable footprint there may be no detection at all. **Objective:** parameter-bound clearance under resource-side enforcement narrows this for governed resources.

Fabricated human authorisation. At every rung currently in force, the named authoriser is a claim the agent makes. At Rung 2 that claim is bound to the agent's key, which establishes that *this registered agent* made it — not that the human did. A compromised agent with a valid signing key can produce a well-formed, correctly signed authorisation for a human who never authorised anything. **The four-eyes structure is enforced; its truthfulness is not established. Objective:** Rung 3.

Not addressed.

Acting without asking. An agent that never submits a request is not governed. Nothing in the rail observes it. §2.2 states the detection boundary and its limits.

Replay within a single action class. The signed form binds the action type and carries a timestamp checked against a permitted skew window. Whether a single valid authorisation can be presented more than once within that window for two distinct actions of the same class is a property of the canonical field set, stated in §8. Any mechanism that depends on the timestamp also depends on the correctness of the server clock, which is not independently attested.

Credential theft between agents. Authentication establishes possession of a credential, not that the intended process holds it. A copied API key or signing key impersonates its owner completely. Credential issuance, rotation and revocation are covered in §10; workload attestation is not built and is not on the near roadmap.

Authenticated request flooding. An agent holding a valid credential is not rate-limited on the clearance endpoint. See §3.5.

3.2 An administrator of the deployment

Capability. Write access to the application directory — **not merely to the policy file, but to every line of code the rail executes.** In the hosted deployment this is OQIRON operations staff; in an on-premise deployment it is the customer's own administrators, who may be the same people who build the agents. The policy file, the application code and the service account are not separated: all are owned by, and writable by, the account the service runs as.

This is the most consequential adversary in the model and the one where the system is weakest.

Prevented.

Silent alteration of past decisions through the application. Evidence rows and checkpoints are protected by a database trigger that refuses UPDATE, DELETE and TRUNCATE, and the application's database role holds only INSERT and SELECT. An administrator with database superuser rights can remove that protection — but see §3.3: they still cannot produce a countersignature.

Detected but not prevented.

Changing the policy to obtain a desired verdict. An administrator can edit the control catalogue, restart the service, have an agent perform an action that is then legitimately cleared under the policy in force, and restore the catalogue afterwards. The evidence chain remains internally sound and correctly records that a permissive policy applied.

Two mechanisms bear on this and neither prevents it. The signed core pin detects a catalogue that differs from the signed digest — **but only when someone runs the verification**, which is a manual act with no schedule. And since 8 August 2026 each change to the deployed catalogue is sealed into the evidence chain, so a loosen-act-restore sequence appears as two policy events bracketing a decision.

The limit of that detection must be stated, because the same administrator controls the detector. The core pin does not cover the programs that compute and verify it, and the policy-change recorder is a file in the same directory with the same ownership. An administrator with code-write capability can disable the recorder, change the policy, act, restore both, and leave no policy event. **Under honest execution of the deployed code the sequence is recorded; the recording is not an independent control against the party who can modify the code.** An independent control would require the recorder to live outside the administrator's modification boundary.

Lowering the assurance requirement. Because the set of action classes requiring a signed assertion is defined in the catalogue itself, and there is no separate minimum-rung configuration, an administrator who can change the catalogue can move an action class from Rung 2 to Rung 1 — with the same detection

properties and the same limit.

Not addressed.

Modification of the decision logic itself. This is the capability the section title implies and it deserves naming separately. An administrator with application-write access can alter authentication, tenant derivation, signature verification, the evaluation of any control, the construction of the evidence record, and the sealing path. "An agent cannot suppress a control" remains true while the administrator removes the control from the evaluator. **Nothing independently attests that the code executing today is the code OQIRON intended to deploy.** There is no code signing, no measured deployment, no immutable artifact deployment, and no separation between the authority to deploy and the authority to operate. The offline anchor key protects a historical commitment; it does not make malicious application code execute honestly. **Objective:** deployment integrity attestation and separation of deployment authority. Not started.

Approval of policy change. There is no maker-checker, no role separation and no approval workflow for changes to the catalogue. A single administrator acts alone. **Objective:** enforced dual authorisation. Not started.

Separation of duty on the deployment. The policy file, the application code and the service account are not separated. This is a straightforward hardening item, it is not done, and §11 records it.

3.3 A database superuser

Capability. Full control of the PostgreSQL instance holding the evidence chain, including the ability to remove the append-only protections and modify or delete any row.

Prevented.

Undetectable forgery of anchored history. This is the case the offline key exists for. A superuser can remove the immutability protection, rewrite evidence rows, delete records and rebuild the hash links so that the chain is internally consistent. They cannot produce a valid countersignature over the result, because the private key is not on the machine and never has been. Any verification against an independently obtained public key computes a different tip than the signed checkpoint asserts and fails.

Detected but not prevented.

Destruction. A superuser can delete evidence. Nothing recovers it except backups, and a verifier presented with a shortened record reports what it can — see the truncation case in §3.7, which bounds what "reports" means.

Rewriting the unsigned tail. Records added since the most recent countersigned checkpoint are protected by hash linkage alone. An adversary who rewrites the entire tail consistently produces a record that verifies internally and is covered by no signature. The published verifier reports the size of the uncovered tail and raises an explicit warning above ten percent. **That threshold is a presentational aid, not a security boundary** — a proportion of chain size does not map to tamper exposure, and the security-relevant quantities are the absolute number of unsigned records, the age of the oldest, and the elapsed time since the last checkpoint. §9 states what the tool reports.

Anchoring a rewritten tail. The tamper window has an exit the previous paragraph does not close. An adversary who rewrites the unsigned tail and waits for the routine anchor ceremony may have the manipulation countersigned, because **the staging process computes the tip from the same database the adversary controls, and the operator is presented with a sequence number and a digest with nothing to check them against.** §7 states what the ceremony verifies today and what it would need to verify to close this. This is a recognised gap, not a solved problem.

Not addressed.

Confidentiality of evidence at rest. A database superuser reads everything the chain contains, including the payloads of disclosed records. The evidence guarantee is integrity, not secrecy.

3.4 The operator of the rail — OQIRON

Capability. Everything in §3.2 and §3.3, plus custody of the offline anchor private key.

A reader evaluating shared infrastructure should ask this directly, so we state it before it is asked: **what stops OQIRON from rewriting the record and countersigning the replacement?**

Not prevented by the cryptography. OQIRON holds the private key. An operator willing to rewrite history can rewrite it, recompute the chain, and sign the result. Every mechanism in §3.3 that defeats a database superuser is available to the key holder as a way of making a rewrite verify. This is inherent: **a single-custodian signing scheme cannot defend against its own custodian.**

What does constrain it, and it is not nothing. The constraint is external retention. A countersigned checkpoint is a small artifact — a digest and a signature. **Once a counterparty, regulator or auditor holds a copy of a checkpoint obtained at a given time, OQIRON cannot rewrite any record covered by it without that holder's copy disagreeing.** The rewrite produces a chain whose tip does not match the retained checkpoint, and the discrepancy is detectable by anyone with both. The published verifier accepts an externally supplied key precisely so that this comparison does not depend on OQIRON's cooperation.

As built, this constraint is available but not systematised. There is no scheduled distribution of checkpoints to independent holders, no external witness, and no third-party timestamping. A recipient who retains what they are given is protected for the period they have retained; **a recipient who retains nothing is trusting OQIRON.** This is the precise sense in which R4 is conditional, as §1.2 states.

Objective. Routine publication of checkpoints to independent holders, so that the record's integrity does not rest on the custodian's continued honesty. Distribution mechanism not yet designed. §11 records it, and a reader for whom operator honesty is not an acceptable assumption should weigh it accordingly — particularly for an on-premise deployment, where the same reasoning applies to the customer's own custodian.

3.5 An external attacker

Capability. Network access to the public endpoints, no valid credential.

Prevented.

Unauthenticated clearance. Every clearance request requires a credential. A request without one is refused before evaluation. An API key lacking the required scope is refused with a distinct response; **a bearer token is not scope-checked here at all**, so scope is not what prevents this and must not be read as though it were. See 11.2.23.

Unauthenticated evidence access. Every route returning evidence data requires authentication, with a single exception: one endpoint returns a global count of governed actions with no credential. It carries no tenant-attributable, personal or commercially specific data, but it is pollable, and an unauthenticated observer sampling it learns platform-wide activity timing and volume. §10 states the intended remedy.

Transport interception. TLS terminates at the reverse proxy; the application listens only on loopback, as do the database and the sibling services on the same host.

Credential recovery from the store. Human passwords are stored with bcrypt at cost 12 over a normalising pre-hash chosen so that no password is truncated by bcrypt's length limit. §10 states the construction precisely.

Detected but not prevented.

Authenticated request flooding. Until 10 August 2026 the clearance endpoint carried no rate limit, and a caller holding a valid credential could occupy the rail. **A per-credential rate and concurrency limit is now in force**, keyed on the credential rather than the source address, leaving at least half the capacity available to other callers.

The residual is capacity rather than admission. **Four synchronous workers are the entire clearance capacity** — the connection pool is nowhere near binding, and workers are what saturate. Beyond four requests in flight the remainder queue until a timeout fires, and no scheme reserves capacity per tenant. Combined with the fail-closed contract (§3.6), sustained pressure still degrades governed action for every caller, and the limit raises the cost of causing it rather than removing the possibility. §11 records the residual.

Not addressed.

API key storage. API keys are stored as unsalted SHA-256 digests. For high-entropy machine-generated keys this is materially different from password storage and is not exposed to dictionary attack; §10 states the generator and its entropy so the argument is testable rather than asserted. It remains weaker than a keyed construction and §11 records it.

3.6 The availability adversary

Capability. Any means of making the rail unreachable — network, resource exhaustion, or an outage with no adversary at all.

This is a consequence of the security design rather than a defect in it, and it belongs in the threat model because a partner will otherwise discover it in production.

By design. Every failure mode resolves to a non-verdict the caller must treat as "do not proceed": an unreachable rail, a timeout, a malformed response, a saturated connection pool, and a failure to seal the evidence. There is no configuration in which the client proceeds on failure.

The consequence. Rail unavailability is unavailability of every governed action. An adversary who cannot defeat authorisation can still stop the business by attacking availability, and an ordinary outage has the same effect. The rail is availability-critical infrastructure and should be deployed as such.

As built, the deployment is a single-region, single-instance topology with a bounded worker count and a connection pool that fails closed on saturation; §10 states the specifics. There is no active-active configuration, no local decision cache, and no degraded mode.

There is deliberately no fail-open path. A break-glass mechanism can be strongly authenticated, dual-controlled, time-limited and scoped — the objection is not that such a design is impossible. It is that any bypass weakens the invariant that a governed action requires a rail verdict, and that invariant is what every claim in this document rests on. Institutions requiring emergency continuity should address it through availability engineering rather than through an exception to the clearance contract.

Objective. Resilience appropriate to the criticality: redundancy, regional failover, and a defined recovery posture. Not built.

3.7 A dishonest party assembling an evidence bundle

Capability. Control over what is included when producing a bundle for a third party. This adversary is not necessarily OQIRON; in a customer deployment it is the institution being examined.

Prevented.

Substitution. Each withheld record is represented by its fingerprint, which is a binding commitment to its original content. A different version cannot later be produced that matches it. If the true record surfaces — by subpoena, from a counterparty, or from another copy — the substitution is evident.

Concealed removal or reordering within the range supplied. Every record participates in the chain whether or not its content is disclosed, so a record cannot be dropped from the middle or moved without breaking the linkage.

Detected but not prevented.

Selective disclosure. The assembler chooses which records to disclose in full. The verifier cannot distinguish a record withheld for confidentiality from one withheld because its contents are unfavourable. It reports the count and proportion of withheld records prominently, and states that a withheld record is unexamined rather than verified. The boundary is favourable and worth stating precisely: **withholding permits concealment, not falsification.**

Truncation at an authentic earlier checkpoint. This is the sharpest attack on the evidence model and it requires no cryptography at all. A party supplies records up to an earlier checkpoint, that checkpoint's genuine signature, and nothing after. Every record is authentic, every link recomputes, the signature

verifies. **A hash chain cannot establish that the record continues beyond the point shown** — a truncated bundle is, from the inside, indistinguishable from a complete one.

Two things bound it. The bundle carries its own summary of what it should contain, and the verifier cross-checks four fields including the recomputed tip; a party who truncates without also making the summary self-consistent produces a file that **contradicts itself**, which the tool reports. And a party who does make it consistent still cannot defeat the second measure, which is that the tool no longer claims completeness: it states the date and coverage of the most recent signature *in the file* and says plainly that whether the record continues beyond it cannot be determined from the file.

The remedy is not cryptographic. A recipient must obtain the date and coverage of the latest checkpoint through a channel independent of the bundle — the same channel already required for the public key — or retain a checkpoint of their own before the period in dispute. §9 states what the tool reports and §11 records the limit.

Selective non-disclosure of a governed action's existence. An action the rail never saw (§2.2) is not in the chain to be withheld.

3.8 Compromise of the anchor signing capability

Capability. Access to the offline private key, the machine that holds it, or the ceremony that uses it — by theft of the medium, compromise of the signing workstation, coercion of the signer, or an unauthorised signature obtained by other means.

This adversary is distinct from a dishonest operator (§3.4) and must be modelled separately, because §3.0 names the key's isolation as the principal cryptographic boundary. A boundary that is not threatened has not been examined.

Prevented.

Extraction from the production environment. The key has never been present on any OQIRON server. Compromise of the rail's hosts does not yield it.

Not addressed.

Theft or compromise of the signing machine. An adversary in possession of the private key can countersign an arbitrary chain state. Every guarantee in §3.3 and §3.4 falls to this adversary, and no mechanism in the deployed system detects it.

Detection of unauthorised signing. There is no record, external to the signer's own machine, of which checkpoints were legitimately signed. A signature produced by a thief is indistinguishable from one produced by the custodian.

Revocation and recovery. There is no defined process for revoking a compromised anchor key, no key rotation procedure, and no statement of what happens to the evidentiary standing of records anchored before a compromise is discovered. **This is a material gap.** §7 states the current custody arrangement; the lifecycle — rotation, revocation, compromise response, and re-establishment of trust in historical records — is not designed. **Objective.** §11 records it.

Verification of what is being signed. §3.3 notes that the operator is presented with a digest computed from the production database. A signer who cannot independently corroborate what they are signing cannot detect that they are being asked to certify a manipulated state.

3.9 Out of scope

Correctness of the agent's reasoning. §1.6.

Truth of what was declared. §1.6.

The security of the systems the agent acts upon. MURAQIB governs the decision to act. Hardening of the payment interface, the mail path or the database is the institution's own responsibility. **Under resource-side enforcement this boundary moves**: OQIRON-supplied enforcement components would become part of the authorisation boundary and require their own threat model (§2.1).

Physical and supply-chain security of the host. Compromise of the operating system, the language runtime, or a dependency is outside the model. The precise guarantee that survives such a compromise is narrower than "the record is safe": **evidence covered by a checkpoint that a verifier obtained and retained independently remains detectably immutable against subsequent compromise of production storage.** Evidence not yet checkpointed, and future execution of the rail, are not protected by the anchor key at all.

Confidentiality of evidence content. The chain provides integrity. Encryption at rest, access control over evidence reads, and tenant read isolation are covered in §10. **Tenant read scoping is presently not enforced**; §10 states what that permits and §11 records it.

3.10 Summary of the residual risks

Five that should determine a deployment decision:

A determined caller can bypass the rail entirely. No enforcement point exists. Detection depends on effect-side correlation the institution performs itself, and some actions leave no correlatable effect.

An administrator can change both the policy and the code that enforces it. Changes are recorded, not approved; and the recording mechanism is within the same modification boundary. Nothing independently attests the deployed code.

The evidence guarantee rests on a single custodian, and that custodian's key has no defined lifecycle. Cryptography defeats a compromised server; it does not defeat the key holder, and there is no rotation, revocation or compromise-recovery procedure. External retention of checkpoints is the available mitigation and is not yet systematised.

Four-eyes authorisation is structurally enforced and not independently established. The named human's participation is asserted by the agent, cryptographically bound to the agent at Rung 2 for five action classes, and independently verified at no rung currently in force.

The rail is availability-critical and singly deployed. Fail-closed means an outage stops governed action, the clearance endpoint is not rate-limited, and there is no redundancy or degraded mode.

4. The clearance path

As built throughout. This section traces a clearance request from arrival to sealed verdict, in the order the code executes. Where a stage does something other than what its name suggests, that is stated.

4.1 The request

A caller submits a declaration of an action it intends to take: an action type drawn from a published taxonomy, a human-readable summary, an identifier for the human on whose behalf it acts, and a set of declared attributes — whether the action is external, whether it carries an attachment, a data-sensitivity classification, a priority, a financial amount, and, where the action requires four-eyes authorisation, the identity of the authoriser and the makers.

The rail evaluates the declaration, not the action. This is the single most important property of the interface: the caller says what it intends to do, and the verdict addresses what it said.

The summary is bounded at 16,384 characters. An over-length summary is **refused, not truncated**, because the summary is sealed into the evidence record verbatim and a shortened one would not match what the caller declared. A request body above 1 MiB is refused before the handler runs.

4.2 Authentication and identity binding

Three credential mechanisms are accepted, tried in order: a bearer token issued by the rail's token endpoint, an API key presented in a request header, or an authenticated dashboard session. A request carrying none is refused before any evaluation occurs. An API key that does not hold the scope required for the clearance endpoint is refused with a distinct response, so a scope failure is not confused with an authentication failure.

The order above is also the limit of that scope check. Because the bearer token is tried first, a request carrying one never reaches the scope test, and the token endpoint issues a token to any valid key without reading its scopes. A credential recorded as holding `read` alone can therefore clear actions by exchanging itself for a token first. This is recorded at 11.2.23 and was demonstrated on 13 September 2026. **A reader must not take a credential's scope list as a statement of what it can do.**

The agent identity used for evaluation and sealed into the record is taken from the credential, never from the request body. The body may contain an agent identifier; it is compared, and both the claimed value and the mismatch are recorded, but it does not participate in the decision.

Three properties of this stage are stated because a reader will otherwise assume more than is true.

Precedence, not conflict detection. A request presenting more than one credential is resolved by order: a valid bearer token short-circuits the API key path entirely, and the second credential is never read or validated. **An invalid bearer token falls through to the API key**, and the rejected bearer leaves no record that it was presented. A request could therefore authenticate as one principal having also offered a rejected credential for another, and the evidence record would show only the successful one. §11 records this.

API keys are accepted in a request header only. A query-parameter fallback existed until 9 August 2026 and was removed: a URL carrying a credential is retained in reverse-proxy access logs, their rotated archives, and backups. No occurrence was found in the retained log window and no caller used it.

A dashboard session authenticates as the human, and the rail treats that as an unregistered agent. The user identifier is sealed as the acting identity, the request is marked as coming from an unregistered principal, and a control fires to escalate it. The path is governed rather than silently permitted. No sealed record has ever taken it. Cross-site protection on that path rests on the session cookie's same-site policy alone; there is no CSRF token and no origin check, which §11 records.

4.3 Context assembly and server-derived signals

The declared fields are assembled into an evaluation context. Several values are then **overwritten by values the server derives**:

- **Agent identity** is replaced with the authenticated identity.
- **Tenant** is resolved server-side from the authenticated agent. A client-supplied tenant is ignored and the attempt logged.
- **Environment** — production or sandbox — is resolved from the agent's registration. This determines which evidence lineage the record is written to, and it is the one derivation that **fails closed by refusing the request**: if the environment cannot be resolved, the rail returns a non-verdict before anything is written, rather than guessing.
- **Risk score** is computed from the declared attributes. §4.10 states what it does and does not affect.

4.4 Authorisation assertion verification

Where the request carries four-eyes fields and the calling agent has a registered public key, the assertion is verified. The agent signs a canonical serialisation of the authorisation claim; the rail recomputes that serialisation and verifies the signature against the registered key.

Verification and requirement are different things, and the distinction matters to anyone reading an evidence record. Verification runs whenever both preconditions hold — registered key, authoriser supplied — regardless of action type. The *result* affects the verdict only for the five action classes that require Rung 2; outside those classes an invalid signature is recorded and changes nothing.

A consequence a consumer of evidence must understand: the signature field has **three distinguishable states** in the record — absent, where verification was never attempted because the agent had no registered key or no authoriser was supplied; present but null, where verification was entered but no result set; and an explicit true or false. Of 433 sealed records, 198 carry no signature field at all, 203 carry it as null, 27 as verified, and 5 as failed. **A reader who interprets absence as "not signed" and a reader who interprets it as "a signature was not required here" will reach opposite conclusions from the same record, and the record does not disambiguate them.** §11 records this.

Verification failures are reported as distinct reasons — unknown key, missing signature, stale timestamp, invalid signature, unsupported binding version — so that a diagnostic failure cannot be mistaken for a cryptographic one.

What the signature establishes is stated in §2.4 and repeated in one sentence because it is the claim most easily over-read: it establishes that the registered agent made the assertion, not that the named human authorised anything.

4.5 Policy evaluation

The assembled context is passed to the control catalogue. Every applicable control is evaluated and the strictest verdict wins; §5 describes the catalogue and the resolution rule.

An action type the catalogue does not recognise does not fall through to approval. Before the catalogue is consulted at all, an unrecognised action type seeds an escalation, so the resolution starts from "a human decides" and can only be made stricter. §5.4 states precisely where that mechanism lives, since it is not one of the 58 controls.

4.6 Declared-summary scanning

This is not inspection of the action itself. The scan reads three declared fields — the summary, the actor's name, and the actor's identifier — for personal data. It does not read an outbound message, an attachment, an exported file, a database payload, or the signed authorisation body. A summary reading "monthly report" attached to an action that exports a customer list scans clean.

A finding at the highest severity elevates an otherwise-approved verdict, so this stage can change the outcome. The scan result is recorded alongside the decision.

Failure behaviour. The scan has no error handling of its own. An exception propagates out of the request before any evidence is written: the effect is fail-closed — no clearance, no seal — but the caller receives an unstructured server error rather than the governed non-verdict, and nothing is recorded. There is no "could not classify" state: content the scanner does not recognise returns clean, indistinguishable from a genuine clean result. Worst-case scan cost under the summary cap is measured at under nine milliseconds. §11 records the unstructured failure.

4.7 Sealing

The verdict, the request as evaluated, the control that determined it, the derived signals, and the digest of the policy catalogue that judged it are written into the evidence record and sealed into the hash chain **in a single database transaction**.

This is a correction, and §6 gives the detail. Until 8 August 2026 the evidence write and the chain seal were separate transactions, and a failure of the seal was deliberately swallowed so the response was never blocked — with the consequence that a caller could receive a verdict the chain did not contain. One such verdict exists in the production record and is disclosed in §11. **As built today, if the seal fails the request fails: the transaction rolls back and the caller receives a non-verdict.** A verdict that was returned was sealed.

The policy digest is new as of 9 August 2026 and is discussed in §5.6. Its cost is a namespace lookup: the digest is computed once per worker process at startup and read from memory thereafter. Because it is computed per worker and workers recycle, two records sealed moments apart during a rollout

may carry different digests — both truthfully, each naming the catalogue that particular worker ran.

4.8 What the rail returns — and what the caller must do

The response carries a decision of `APPROVED`, `BLOCKED` or `ESCALATED`, the control that determined it, an explanation in English and Arabic, the derived risk score and level, and the evidence identifier.

Two responses are **not** verdicts and must not be treated as any: `REFUSED`, returned when the environment could not be resolved, and `UNAVAILABLE`, returned when the connection pool is saturated or the seal could not be written. The client library resolves every failure mode — unreachable rail, timeout, malformed response, missing credential, signing failure — to the same non-verdict, and states that there is no configuration in which a caller proceeds on failure.

`ESCALATED` is terminal at the interface. There is no endpoint that converts it to an approval, no callback, and no polling path. Escalation means the rail will not clear this action; the institution's response is a human process outside the rail. Where an institution operates a hold-and-authorise flow, the authorised action returns as a **new** request of a distinct action type — one of the five requiring a signed assertion — evaluated on its own merits. The rail holds no state waiting for a human.

Fifteen declared attributes are recorded and do not affect the verdict. They are evidence, not inputs. §5.2 states what the controls do read, because the difference between the two lists is the substance of what the catalogue governs.

4.9 What the caller controls, and what it does not

The caller controls	The rail derives
Action type	Agent identity
Summary and actor fields	Tenant
Declared attributes	Environment
Financial amount	Risk score and level
Authoriser and maker identities	Verdict and control determined
Whether a signature is supplied	Whether one is required
	Policy digest, evidence identifier, chain position

The rail does not independently establish the real-world truth of anything in the left column. Some of those fields are validated in narrower ways — an action type is recognised or it is not, a summary is scanned for personal data, an authorisation assertion may be cryptographically bound to the caller — but none of that establishes that the declaration matches what the agent went on to do. The right column is what the rail knows independently of what it was told.

4.10 The risk score

As built, the risk score is a recorded observation with no authority over the verdict. It is computed from fourteen weighted signals over the declared attributes, banded, sealed, and used in dashboard aggregates. **No control in the catalogue reads it.** This was verified by sweeping every recognised action type against every combination of declared flags across all risk bands including the score's absence: no combination changed either the verdict or the control determined.

This is stated plainly because the alternative is a reader assuming a CRITICAL risk score gates something. It does not. Whether it should is an open design question in §11, and until it is resolved the score should be presented in any interface as an observation rather than as a control outcome.

Two properties, corrected on 8 August 2026 and disclosed rather than quietly fixed:

A signal that cannot be evaluated is now labelled, not dropped. Previously a signal whose input could not be interpreted was silently skipped and its weight omitted, producing a score lower than it should be with nothing to indicate why. The scorer now reports the composite from what evaluated, sets the band conservatively to what it would reach if every unevaluated signal fired, and marks the score incomplete with the signals named. A score and a band may therefore legitimately disagree; where that occurs the record says so.

A weight that was recorded but never applied has been connected. A regulatory-risk flag was sealed into records under one name and read by the scorer under another, so it contributed nothing. Eleven sealed records carry that flag and a score that does not reflect it. Sealed records are immutable and have not been altered; records sealed from the correction onward carry the weight. §11 records both facts, and §8 records the schema-versioning obligation this exposed.

---

5. The control catalogue

As built throughout.

5.1 Shape

The catalogue holds **58 controls** across 28 dimensions — among them market conduct, agent authority, financial crime, data protection, external communications, model risk, and the clearance lifecycle itself. Controls sit in three tiers: a small set governing the agent's own standing, a larger set governing action classes, and a set derived from regulatory expectations.

Two controls are gated by jurisdiction. The jurisdiction is read from a deployment-level setting outside the catalogue; §5.7 states what that means for integrity.

5.2 What the controls actually read

A control count is a poor proxy for policy depth, and §4.8 has already said that fifteen declared attributes do not affect the verdict. The two statements need reconciling, so the inputs are stated directly.

The controls read, in order of how many depend on them: **the action type**, which selects most of the catalogue; **the agent's standing** — whether it is frozen, unregistered, operating autonomously, exceeding its registered scope, or under an emergency freeze; **the authorisation structure** — whether an authoriser is named, holds a permitted role, is distinct from the makers, and whether the assertion was signed and verified; and **a small number of server-derived conditions** including clearance state, off-hours activity, velocity, and recipient restriction.

They do **not** read the financial amount, the data-sensitivity classification, the externality flag, the attachment flag, the priority, or the regulatory-risk flag. A transfer of a hundred riyals and a transfer of a hundred million receive the same treatment from the catalogue; what distinguishes them, if anything, is the action type the caller declared.

This is a real property of the current design and it should be read as such rather than inferred from the control count. The catalogue governs *what class of action an agent may take, in what standing, under what authorisation* — not the magnitude or sensitivity of a particular instance. Whether magnitude should be a policy input is an open design question recorded in §11; the declared attributes are sealed into evidence today so that it can become one without losing history.

5.3 Selection and resolution

A control declares either the action types it applies to, or a predicate of its own, or neither. A control with declared triggers evaluates when the request's action type is among them. A control with neither evaluates on every request and gates inside its own decision function — this is how agent-standing controls work, since "is this agent frozen" applies regardless of what was asked.

Every applicable control returns a verdict or abstains. The **strictest verdict wins**: BLOCK outranks ESCALATE, which outranks APPROVE. There is no weighting, no voting, no override, and no path by which a permissive control overrides a restrictive one. A single BLOCK blocks the action regardless of the other fifty-six.

Where several controls return the same strictest verdict, the record names the first in catalogue order. This is deterministic for a fixed catalogue, and it carries no meaning: reordering the catalogue would change which control is named without changing any verdict. Reordering would, however, change the pinned digest, so it cannot happen unobserved.

5.4 The decision pipeline — the catalogue is not the whole of it

A reader asking "what can cause a BLOCK or an ESCALATE?" cannot answer it from the catalogue alone, and this document should not imply otherwise. Verdict formation has three parts:

An unrecognised action type seeds an escalation before evaluation begins. This is not one of the 58 controls; it is a step in the evaluation function itself, recorded under its own identifier. It is nonetheless covered by the signed pin, because the pin hashes the whole evaluation file and not only the per-control entries. **It is also the most frequently recorded determining control in the chain: 152 of 433 sealed records — 35% — name it.** The most-cited control in the evidence record is not a member of the catalogue, and a reader inspecting the 58 will not find it there.

The catalogue then evaluates, as described above.

Declared-summary scanning can then elevate an approval (§4.6). This too sits outside the catalogue.

The 58-control catalogue is the **primary policy-evaluation authority**, not the sole authority. Signing the catalogue signs the largest part of the decision logic and not all of it; the whole-file pin covers the first mechanism, and §11 records the scanner's rule set as outside the pinned artifact.

5.5 The registry that is not the policy

The rail exposes a policy registry containing a larger set of policy records retained from an earlier generation of the system. **It is not consulted at decision time.** An administrator editing something labelled "policy" there changes nothing about authorisation. This is an operational hazard rather than a security control, and it should be removed or unmistakably relabelled rather than left to a footnote; §11 records it.

5.6 Policy integrity and provenance

What the catalogue is, is verifiable. A signed pin binds the deployed catalogue to a digest countersigned with a key held offline. The digest covers the raw bytes of both catalogue files, the control count, and per control: identifier, dimension, tier, declared decision, jurisdiction gate, triggers, and a hash of the source of its decision function.

Proven by failure, not by absence of error. A single altered verdict inside one control's decision function — invisible to the control count and to the control's declared metadata — is detected and the control is named. The verifier has been run against a deliberately corrupted copy and refused it.

Since 9 August 2026, every evidence record carries the digest of the catalogue that judged it. This is the change that converts policy provenance from inference into direct binding. A reader holding an evidence record can ask whether its stated digest corresponds to a catalogue that was signed, without reconstructing state from a separate sequence of change events.

Since 8 August 2026, a change to the deployed catalogue is itself sealed into the evidence chain, recording the digest before and after, the control count, whether the deployed catalogue matched a valid offline signature at the time, and the declared identity and reason of whoever performed the ceremony. A change detected at startup that nobody declared is recorded as such with the identity field empty, because the system cannot know who made it and will not invent an answer.

The first such record is sealed at chain sequence 457 and is marked retroactive. **Policy history therefore begins there.** For decisions sealed from that point, the chain records catalogue-state transitions and each record names its own digest. For the 433 decisions sealed before it, neither is available: their records carry no digest, and the catalogue changes that preceded them are not in the chain and cannot be added, since the records that would have carried them are sealed. **An auditor can establish which controls were in force for a decision sealed after 9 August 2026. For earlier decisions they cannot.**

What none of this earns is that a change was authorised. There is no approval, no maker-checker, and no role separation over policy content. The declared identity and reason in a ceremony record are supplied by the person performing it — an assertion, recorded faithfully, not an independently established fact. And per §3.2, the recording mechanism sits within the same modification boundary as the code it records. **MURAQIB cannot establish who, if anyone, approved a policy change, because it has no policy-change authorisation mechanism.** A change may have been approved by any process outside the system; the system knows nothing of it.

5.7 Two limits on what the record can tell an examiner

Granularity. For a change to a control's own definition — its decision function, its triggers, its declared verdict — the chain record names the control and the field. For a change to a structure the catalogue shares across controls, the record states only that the catalogue's bytes changed. The correction that brought a fifth action class under the signature requirement is of the second kind: it altered a lookup structure rather than any control's definition, so the record states that the file changed and not which rule moved.

Jurisdiction. The jurisdiction setting that gates two controls is read from a deployment-level environment value. It is **not covered by the signed pin, not sealed into any record, and not logged when changed** — the mechanism that consumes it is pinned, but the value is not. Today only one jurisdiction is defined and every unrecognised value falls back to it, so changing the setting changes nothing; that is an accident of there being a single binding set, not a protection. The moment a second is defined, an unsealed and unlogged deployment value begins deciding which controls run. §11 records it.

And the record cannot replay its own verdict. Of the twelve inputs the controls read, two are sealed. The rest — agent standing, clearance state, off-hours, velocity, recipient restriction and the others — are derived at evaluation time and not written into the record. A sealed record now establishes what was decided, by which catalogue, on what declaration. It does not contain enough for a third party to recompute the decision and confirm it followed. §11 records this as the largest remaining gap in the evidence model.

6. Fail-closed behaviour

As built throughout.

A governance rail is defined less by what it does when things work than by what it does when they do not. This section enumerates every point in the clearance path at which something can fail, and states what actually happens — not what was intended.

6.1 The principle, stated precisely

No failure of an authorisation-critical dependency produces a clearance. Every such error path resolves to a refusal or a non-verdict. There is no configuration, no flag, and no fallback in which the rail approves an action because something went wrong.

The qualifier is load-bearing and not a hedge. One dependency is deliberately **not** authorisation-critical: tenant attribution. When the tenant lookup fails, the rail records a distinct sentinel marking that attribution failed and continues evaluating. This is defensible only if tenant identity cannot influence the decision, and it does not: **no control in the catalogue reads tenant identity, no catalogue or jurisdiction binding is selected by it, and no threshold varies with it.** Tenant is a bookkeeping attribute of the record, not an input to the verdict. Refusing clearance because a bookkeeping lookup failed would be the wrong trade; recording the failure honestly is the right one. One production record carries this sentinel.

If tenant ever becomes a policy input, this path must change with it. §11 records the dependency.

The principle has a price, stated in §3.6 and repeated because it follows directly: an unreachable rail makes every governed action unavailable. The rail is a single point of failure for governed work by design, and that design is defensible only if it is deployed as availability-critical infrastructure. §10 states what is and is not in place.

6.2 The failure table

Failure	What happens	Outcome class
No credential, or an invalid one	Refused before evaluation	Refusal — nothing written
API key lacks the required scope	Refused, distinct response	Refusal — nothing written
Bearer token lacks the required scope	**Not checked — accepted** (11.2.23)	Cleared and sealed
Summary exceeds the length limit	Refused, limit named	Refusal — nothing written
Request body exceeds the size limit	Refused before the handler runs	Refusal — nothing written
Environment cannot be resolved	Non-verdict returned	Non-verdict — nothing written
Tenant lookup fails	**Sentinel recorded, evaluation continues**	**Non-authoritative — proceeds**
The policy engine raises	Non-verdict returned, logged critical	Non-verdict — nothing written
An unrecognised action type	Escalated	**Governance verdict**
Assertion signature invalid, on a class requiring one	Blocked	**Governance verdict**
Personal data found in the declared summary	Escalated	**Governance verdict**
Evidence write fails	Transaction rolled back	Non-verdict — nothing written
Chain seal fails	Transaction rolled back	Non-verdict — nothing written
Connection pool saturated	Non-verdict returned	Non-verdict — nothing written
Declared-summary scan raises	Request fails before any write	Unstructured error — nothing written
Any unhandled exception elsewhere	Generic server error	Unstructured error — nothing written

The three-way split matters more than the binary it replaces. **A governance verdict is a determination the rail made. A non-verdict is a statement that the rail could not determine anything. A refusal is a rejection at the door.** Conflating the first two is the defect corrected on 9 August 2026, described below.

6.3 Infrastructure failure is not a governance verdict

Until 9 August 2026, a failure of the policy engine returned `ESCALATED` — safe, because the caller must not proceed on an escalation, but dishonest, because **an auditor reading the record could not distinguish "policy determined a human must decide" from "policy never ran."** The record carried an escalation with empty rule fields, and a legitimately empty rule field occurs elsewhere.

An evaluator failure now returns the same non-verdict shape used for pool saturation and seal failure, with a distinct critical-severity marker naming the failing stage, and the response says what it is: *this is an infrastructure failure, not a governance decision*. Nothing is sealed, because a non-verdict is not a

decision and must not enter the decision chain.

No sealed record in the production chain is an infrastructure failure wearing a verdict. This was established rather than assumed: one record matched the telltale shape and was investigated and excluded — it predates the clean engine becoming authoritative, and the failure marker appears nowhere in the retained logs.

Two shape inconsistencies remain and are recorded in §11: an environment-resolution failure returns a different non-verdict token than the rest of the rail, and an evidence-write failure returns a bare error carrying the raw exception text with no decision field at all.

6.4 What "fails closed" does not mean

Fail-closed is not the same as well-reported. Two failures return a generic server error rather than the structured non-verdict the client contract defines. The caller is correct either way — the client library resolves any non-success to "do not proceed" — but the evidence record is absent and an operator investigating has less to work with.

Fail-closed does not mean recorded. A request refused before evaluation writes nothing to the chain. **An institution reviewing the chain sees what was cleared and what was refused by policy; it does not see what was rejected at the door.** Authentication failures in particular are security-relevant events that leave no trace in the evidence record. This is why R4 in §1.2 speaks of every request that reaches evaluation rather than every request.

Fail-closed applies to the rail, not to the caller. Nothing here prevents a caller that receives a non-verdict from proceeding anyway. §2.1.

6.5 The seal-atomicity correction

Until 8 August 2026 the evidence write and the chain seal ran in separate transactions, and the seal was wrapped in a handler that discarded any failure so the clearance response would never be blocked. The intent was availability. The consequence was that **a caller could receive a verdict the evidence chain did not contain**, with no signal distinguishing such a record from a sealed one.

This is not hypothetical. **One production verdict was returned without being sealed** — an escalation on 4 August 2026, disclosed by identifier in §11. Seventy-one further instances occurred in the sandbox lineage the same day.

The cause was not the swallowed error. The two writes could not have rolled back together even had the error been raised: the evidence row committed some seventy lines before the seal was attempted. Making the failure loud would have produced a caller correctly told "unavailable" alongside a committed record of a verdict never returned — a different inconsistency, not a fix.

The correction makes the evidence write, the explainability trace and the chain seal commit as one transaction. If the seal fails, all three roll back and the caller receives a non-verdict. Since that correction, every returned verdict has been sealed before the response.

Two findings from that work bear on §7 and are recorded here.

The retry behaviour had to change with the transaction boundary. The chain append previously retried on contention by rolling back and trying again. Under a shared transaction that rollback would discard the pending evidence row, and a later successful attempt would commit a chain event with no evidence behind it — the inverse orphan, worse than the original defect. The append now makes a single attempt and fails closed. §10 states the throughput consequence.

A connection-pool defect had silently disarmed the chain's fork protection. The chain serialises appends with a transaction-scoped advisory lock. Connections returned to the pool by certain paths were left in a mode where every statement commits immediately, and on such a connection a transaction-scoped lock releases the instant it is taken. Two workers could read the same chain tip and both attempt to append. The uniqueness constraint described in §7.3 caught every one, which is why this surfaced as failed appends rather than a forked chain — but **the lock had not been serialising anything, and the guarantee rested on the constraint alone.** The pool now repairs the connection mode on acquisition, and the lock has been measured serialising correctly.

6.6 What has not been engineered

Crash between commit and response. If the transaction commits and the response is lost, the record exists and the caller does not know the outcome. **The rail has no idempotency mechanism**: a retried request is a new request, evaluated afresh and sealed as a second record. Two records may describe one intended action.

Concurrent submission of the same action. Nothing deduplicates. Two identical requests produce two clearances and two records.

Both are unremarkable as a decision point, where a duplicate clearance is a duplicate record. **Both become materially more serious under resource-side enforcement**, where a clearance is a consumable artifact and two clearances for one intended action would authorise two executions. An action-intent identity, defined by the Integration Standard and honoured as an idempotency key, is therefore a prerequisite for that work rather than a refinement of it. §11 records both.

---

7. The evidence chain

As built throughout, except where marked.

7.1 What it is

Every clearance decision is written as an event in an append-only sequence. Each event carries a hash of its own canonical content, and a seal computed from that content hash together with the seal of the event before it. The sequence begins from a fixed public starting value.

The construction is deliberately unremarkable: SHA-256, a canonical JSON serialisation, and a linked list. It requires no distributed consensus, no third party, and no network. Its properties come from three things — the linkage, the append-only enforcement, and the offline countersignature — and each is stated separately below because they fail differently.

The canonical form is published in full. The specification accompanying the verification tool states the coercion rules, key ordering, separators, encoding, and the exact concatenation used for the seal, so an independent implementer can reproduce every hash without access to OQIRON code. This has been done: a from-scratch reimplementation reproduced all content hashes and all seal hashes for the entire production chain.

The serialisation does not follow the standard JSON canonicalisation scheme published as RFC 8785, which predates it. That is a deliberate choice rather than an oversight: scalars are coerced to strings so that the database's own numeric normalisation cannot alter a value on re-read, and the format is now bound into every sealed record, so changing it would invalidate every existing hash. It is fully specified and independently reproduced, which is what interoperability requires; adopting a standard scheme is a question for a future chain lineage, not for this one. §11 records it.

7.2 What the linkage establishes

Because each seal depends on its predecessor, altering, inserting, removing or reordering any event changes every seal that follows. A verifier recomputing the chain from the published starting value detects the change and names the position.

What the linkage alone does not establish is that the chain has been shown in full. A record truncated at any point is internally perfect. §3.7 and §7.6.

7.3 Append-only enforcement

Two mechanisms, both at the storage layer.

Uniqueness constraints on the seal and on the predecessor seal make a fork impossible at the storage layer: two events cannot claim the same predecessor, and no seal can repeat.

A trigger rejects any update, deletion or truncation of the evidence tables. Its function contains no role-based bypass, and the application's own database role holds only insert and select privileges.

Both are database objects, and the scope of what they protect must be stated precisely. While the database's integrity constraints remain in force, no application-level race can persist a fork and no application compromise can alter a record. An operator with schema-level privileges can remove the trigger and the constraints, and then modify rows freely. **What they cannot do is produce a countersignature over the result** — that is §7.4's job, and §7.5 states its limit.

Sequence numbers are not contiguous, and this is normal. Numbers are allocated before an event is confirmed, so a rolled-back transaction consumes a number that never appears; the production chain has twenty-one such gaps. **The chain links by hash, not by number**, so a gap breaks nothing, and the bundle format states this explicitly so an external verifier does not report false breaks. One consequence follows: **sequence continuity cannot serve as an independent completeness check**, which is why the bundle carries a separate committed event count and checkpoint coverage.

7.4 The offline countersignature

Periodically the chain's tip is countersigned with an Ed25519 key **whose private half is held outside the production environment and has never been present on any OQIRON server.** Signing is a manual act performed by a person on a separate machine. The signed digest, the signature and the key identifier are written into a checkpoint record under the same append-only enforcement.

This is what resists an adversary who has taken the servers. They can rewrite events and recompute the linkage consistently; they cannot produce a signature over the result. Verification against the published public key computes a different value than the checkpoint asserts, and fails.

The public key is published at `oqiron.ai/verify`, on a channel separate from any evidence file. This is worth stating precisely, because the protection it offers is narrower than it appears. **Separate publication defends against substitution of an evidence bundle. It does not make verification operator-independent**, because the key and the evidence both originate with OQIRON. A verifier who obtains all trust material from OQIRON after a dispute arises has no independent root by which to distinguish the historical key from a replacement. Operator-independent verification requires a trust anchor outside OQIRON's unilateral control — a fingerprint retained by a regulator, a checkpoint held by a counterparty, or an independent witness. §7.7 states the position, §1.2 states it as R4's condition, and §11 records it.

The custody arrangement has no lifecycle. There is no key rotation procedure, no revocation mechanism, and no defined position on the evidentiary standing of records anchored before a compromise is discovered — how compromise time would be established, how a successor key would be authorised, how a verifier would discover the historical key set, or whether a revoked key still validates historical checkpoints. For infrastructure whose evidentiary lifetime is measured in years this is not a refinement; **it is a prerequisite for regulated deployment** and it is not designed. §3.8 states the adversary; §11 records the gap as material.

One key serves two purposes with no domain separation. The same key countersigns the evidence tip and the policy catalogue digest (§5.6), and both sign a bare hexadecimal digest with no purpose prefix or context string. A signature over one digest does not verify against a different digest — the danger is subtler: **the two protocols occupy the same signed-message namespace, so a signed message carries no cryptographic marker of which purpose it was produced for.** The remedy

is both separate keys and a domain-separating prefix in the signed message. Correcting it means either accepting two signature formats indefinitely or introducing the change with a key rotation, so §11 records it as scheduled with the key-lifecycle work rather than patched.

7.5 The ceremony, and what the signer can actually check

A countersignature is only as good as the state being signed. **As built, the signer is presented with a sequence number and a digest computed from the production database** — the same database an adversary in §3.3 would control. The staging process reads that database, recomputes the tip, and stages it. The three guards applied at insertion — signature validity, tip agreement, no duplicate checkpoint — all re-read the same source, and therefore establish **self-consistency rather than authenticity**.

The distinction this forces is worth naming, because it changes how the offline key should be understood across this document: **OQIRON has offline signing-key custody. It does not have independent state attestation.** The first prevents an intruder forging a signature. The second would prevent a signature being applied to a state the intruder prepared. Only the first is built.

The consequence, stated in §3.3: an adversary who rewrites the unsigned tail and waits for the next routine ceremony may have the manipulation legitimately countersigned.

The nightly encrypted off-host copy is sometimes offered as corroboration. It cannot serve as such today, for two reasons. It is encrypted to a key only the operator's own machine holds, so nothing on the server can compare against it. And **storage independence is not state independence**: a copy generated from compromised production state faithfully preserves the manipulation. It is an off-host retained copy, not an independent witness.

Objective. Three improvements, in ascending order of what they achieve and cost.

Operator anomaly detection. Present the signer with a difference rather than a number: the previously anchored position and digest, the new position, how many events were added, and their date range. An operator told "one hundred and thirty-five new events since 1 August" can notice the institution was closed that week. **This is anomaly detection, not state authentication** — it does nothing against a malicious tail containing a plausible number of plausible events — but it is cheap and it raises the floor.

Independent corroboration. Comparison against a copy the production host cannot alter, or against a checkpoint retained by a counterparty. This is the mechanism that actually addresses the gap.

Continuous signing. Signing per batch at write time removes the unsigned tail entirely, and is the structural answer. **It requires a different key architecture, not a change of cadence** — an offline root certifying an online intermediate held in a hardware module, or a forward-secure scheme. Making the current offline key an online dependency would destroy the isolation that gives it value, and this document does not propose that. None of the three is built.

7.6 The tamper window is adversary-specific

Records added since the most recent countersigned checkpoint are protected by linkage alone. But "the window is the anchoring interval" is true for only one adversary, and stating it without qualification would overclaim:

Adversary	What bounds a rewrite
Storage attacker without the key	The most recent checkpoint
Deployment administrator	The most recent checkpoint, if a verifier retained it independently
Operator holding the signing key	Only a checkpoint already held by an external party
Compromised signing ceremony (§7.5)	Independent pre-signature corroboration — none exists today

The anchoring interval is **operator-driven rather than scheduled**, and intervals in the production chain have run to nearly four days and one hundred and thirty-five events. **A tamper window whose width depends on someone remembering to sign is not yet rail-grade.** A hard operational invariant — a maximum elapsed time, a maximum unanchored count, an alert on breach, and a defined posture when it is exceeded — is not defined. §11 records it.

The verification tool reports the size of the uncovered tail and warns above a tenth of the record. **That threshold is a presentational aid, not a security boundary**: a proportion of chain size does not map to exposure, and the security-relevant quantities are the absolute count of unsigned records, the age of the oldest, and the time since the last checkpoint. §9 states what is reported.

7.7 What the chain does not contain — and the distinction that matters most

Tamper-evident logging and verifiable policy execution are different properties, and MURAQIB has the first.

A sealed record establishes what was decided, on what declaration, by which catalogue — the last of these since 9 August 2026, when the catalogue digest began to be sealed into every record. It does not establish that the verdict was the correct result of applying that catalogue to those inputs. **Of the twelve server-derived conditions the controls read, two are sealed.** Agent standing, clearance state, off-hours, velocity, recipient restriction and the rest are computed at evaluation time and not written. **A third party holding a record cannot recompute the decision and confirm it followed.**

This is the largest gap in the evidence model and it is stated here rather than in §11 alone, because it bounds what every other claim in this document means. It is also, as §2 states, the next capability on the roadmap: sealing the full determinative context — all derived inputs, the evaluator version, the configuration state, and the evaluation trace or a commitment to it — would let an independent party replay a decision and confirm it. **Objective.** Not built.

Two further absences: the chain does not contain requests refused before evaluation (§6.4), and it does not contain policy history before 8 August 2026 (§5.6).

7.8 Scale as at this draft

The production chain holds 436 events across 23 checkpoints, the most recent covering seq 456 against a tip at seq 457; B.6(e) states why a small unanchored tail is expected. Every event verifies: content hashes recompute, seal hashes recompute, and every stored predecessor matches the preceding event's seal. Twenty-one sequence numbers are absent for the reason in §7.3.

These figures are as at the date on the cover and will be stale when read. They are a reference point for a reader running the verification tool, not a claim about the present.

8. Verified principal binding

As built throughout, except where marked.

Four-eyes authorisation — a second, distinct party countersigning a consequential action — is the central governance control this rail exists to enforce. This section states exactly what the rail establishes about that human, and what it does not. It is the most carefully worded section in this document because it is the claim most easily over-read.

8.1 The question this section answers

When an evidence record says that a named person authorised an action, what does that record actually prove?

Three answers are possible, and they correspond to the three rungs described in §2.4. **Rung 2 is in production for five action classes; the platform-wide assurance floor remains Rung 1. The third rung is not built.**

8.2 What is enforced regardless of rung

At every rung the rail evaluates the structural requirements of a four-eyes assertion, independently of the calling system:

- An authoriser must be named.
- The authoriser must hold a role permitted for the action class.
- **The asserted authoriser identifier must not be among the asserted maker identifiers.** A request in which the identifier that initiated the action is also the one authorising it is refused.

These are enforced by the rail, not by the agent, and an agent cannot suppress them. **This is the part of four-eyes that MURAQIB genuinely enforces**, and it is not nothing: a caller that submits an action authorised by its own initiator is refused, and the refusal is sealed.

What it cannot establish is that the identifiers denote people, or two different people. The rail enforces structural separation between asserted identifiers. It has no canonical identity source against which to determine that two identifiers are not the same human under two names, that an identifier corresponds to a real person, or that the asserted role belongs to them. §11 records this.

What these checks operate on is the question §8.4 answers.

8.3 Rung 2 as built

Six of eight production identities hold an Ed25519 key pair whose public half is registered with the rail. For those agents, the authorisation assertion is signed.

The signed form. The agent serialises the authorisation claim into a canonical byte string and signs it. The serialisation binds the action type, the agent identity, the authoriser identity and role, the maker identities, the request reference, and a timestamp. Two binding versions exist: the earlier form, in use by

the agents signing today, and a later one using explicit field separators. §11 records that the later form is defined but not the one in production use.

Verification. The rail recomputes the serialisation from the request as it received it and verifies the signature against the agent's registered public key, before policy evaluation. A timestamp outside a five-minute window is rejected as stale.

Failure reasons are distinct — unknown key, missing signature, stale timestamp, invalid signature, unsupported binding version — so a configuration failure cannot be mistaken for a cryptographic one. This has already proved its worth: a stale key cache in a test harness produced "unknown key" rather than "invalid signature," and the distinction identified the harness fault in one step rather than sending an investigation after a phantom cryptographic failure.

Enforcement. For five action classes — authorised external correspondence, document LLM egress, memory ingest, screening disposition, and campaign recipient authorisation — an assertion that is unsigned, invalid, stale, or signed by an unregistered key causes the action to be **blocked**. It is refused, not downgraded to a lower assurance path.

A signature cannot be replayed across action classes. The signed form includes the action type, so a signature produced for one class does not verify when presented for another. This was demonstrated by substitution rather than assumed.

8.4 The boundary

Rung 2 establishes that the registered agent asserted that a named human authorised this action. It does not establish that the human did.

The rail has no channel to that human. It holds no credential of theirs, receives nothing signed by them, and has no means of asking them. The authoriser's identity, role, and distinctness from the makers are values the calling system places in the request and signs with **its own** key. A compromised agent holding a valid signing key can produce a well-formed, correctly signed, structurally valid authorisation for a person who authorised nothing.

At Rungs 1 and 2 the rail enforces the structural requirements of a four-eyes assertion; it does not independently establish human authorisation. The phrase "verified human four-eyes authorisation" should not be read into anything below Rung 3.

What the signature does add over Rung 1 is genuine and should not be understated: it establishes that the assertion originated from the registered agent and was not altered in transit or fabricated by a third party who did not hold that agent's key. It moves the trust from "whoever could reach the endpoint" to "the registered agent." It does not move it to the human.

8.5 Coverage, and the floor

Rung 2 is enforced for five action classes. Every other action type operates at Rung 1, where the authoriser fields are evaluated as supplied. A signature, where one is present on such a request, is verified and its result recorded, but it does not alter the verdict.

The as-built security floor is therefore Rung 1, and this should be read plainly rather than inferred: for the great majority of action types, including financial transfer, the authoriser is whoever the calling system says it is.

Six of eight production identities hold a key. *Identities, not agents in service: as of 13 September 2026 seven of the eight hold no API credential and cannot transact until one is issued (11.2.26). The signing-key registration is unaffected by that and is stated here as it stands.* The remaining two — the legacy identities MAJLIS-002 and MASAAR-001 — cannot use the Rung 2 classes at all — an unsigned assertion on those classes is blocked, so a keyless agent is refused rather than admitted at a lower assurance. A ninth active agent is a sandbox identity and is excluded from this figure, since sandbox records are never part of the production chain; three further registrations are revoked, and twelve are registered in total.

There is no configuration mechanism to require a minimum assurance rung for an action class. Coverage is a property of the catalogue and changes only by changing the catalogue — which, per §1.3, is subject to no approval or role separation. **The two limitations compound: an administrator who can edit the catalogue can lower the assurance required for an action class.** §11 records them together.

8.6 What an evidence consumer sees, and how to misread it

The signature outcome appears in a sealed record in **three distinguishable states**, and the distinction is not documented anywhere a consumer would find it except here.

Absent. No signature field at all. Verification was never attempted, because the agent has no registered key or the request carried no authoriser. 198 of 433 sealed records.

Present but null. Verification was entered but no result was set. 203 records.

Explicitly true or false. 27 verified, 5 failed — four for an invalid signature, one for a stale timestamp.

A reader who interprets absence as "not signed" and a reader who interprets it as "a signature was not required here" reach opposite conclusions from the same record, and the record does not disambiguate them. This is a defect in the evidence format rather than in the enforcement, and §11 records it.

8.7 What is not established at any rung in force

Replay within a single action class. The signed form binds the action type and carries a timestamp checked against a five-minute window. Whether a single valid authorisation can be presented more than once within that window, for two distinct actions of the same class, depends on whether the signed field set includes a value unique to the action. The request reference is included; whether callers populate it uniquely per action is a property of the caller, not of the rail. **The rail does not maintain a replay cache and does not reject a previously seen assertion.** §11 records it.

Trusted time. Staleness rejection depends on the server's clock. Nothing independently attests it. An adversary who can move the clock can widen or narrow the acceptance window.

Key lifecycle for agent signing keys. Registration is by database record. Rotation has been performed — an agent's key was replaced during this programme and the chain shows verification under the old key up to one date and the new key thereafter — but the procedure is manual and unwritten, and there is no revocation mechanism distinct from deregistration. This is a smaller version of the anchor-key gap in §7.4 and §11 records it.

A dishonest institution. Nothing here addresses an institution whose own people genuinely authorise something they should not. Four-eyes is a control against unilateral action, not against collusion, and no cryptographic mechanism changes that.

8.8 Rung 3

Objective. The authorising human authenticates directly to the rail through federated identity, so that the authorisation binds to a credential the human holds rather than to an assertion an agent makes about them. This is the rung at which "the human authorised it" becomes a claim the rail can make.

Not started. A prerequisite exists and is unmet: the coordinating agent's own single-sign-on integration is presently a stub over password authentication, and federated human identity cannot be built on it.

The evaluation logic is identical at every rung — only the provenance of the authoriser fields hardens. But Rung 3 introduces a human authentication flow that does not exist today, so it is a compatibility objective rather than a guarantee that no integration work is required.

---

9. Independent verification

As built throughout, except where marked. This section describes the tool by which a third party checks an evidence record without trusting OQIRON, and states precisely what a passing result does and does not prove.

9.1 Why it exists

Every claim in §7 is worth exactly as much as an outside party's ability to check it. A record that only its custodian can verify is a record whose integrity rests on the custodian's word — which is the position this whole document argues against.

The verification tool is published under the MIT licence at `github.com/oqiron/muraqib-verify`. It is a single file of standard Python: no installation, no dependencies, no network access, no database access, and no import of any OQIRON code. It reimplements the canonicalisation from the published specification rather than sharing it, because a verifier that shared code with the system it verifies would prove considerably less.

The stdlib-only constraint is deliberate and was met without exception, including signature verification. It exists because the people most likely to need this tool — regulators, auditors, institutional security teams — frequently cannot install software on their machines.

9.2 What it checks

Given an evidence bundle, the tool recomputes rather than accepts:

- Every disclosed record's content hash, from the record's own contents.
- The seal chain from the published starting value, advancing on the **recomputed** seal at each step rather than the one stated in the file. Trusting the stated value would allow a bundle to validate a broken chain.
- Every checkpoint signature, against a public key.
- The seal at the point the most recent checkpoint claims to cover, against that checkpoint's signed digest.

Withheld records participate in the chain walk. Their content hash is taken as given, but it must still fit the links on either side, so withholding content cannot conceal a reordering.

It self-tests before it reports. The obvious attack on any verifier is to make its signature check always return true, so the tool verifies known-good signatures and four kinds of deliberately broken signature before touching the bundle. If that self-test fails it produces no verdict at all and says so: *do not trust any result from this program*. This was demonstrated by sabotaging the signature routine and observing the refusal.

9.3 What a passing result proves

That the records present form an unbroken cryptographic chain, and that the portion covered by a signature matches a commitment made under the key supplied.

The tool no longer describes records as *genuine*, and the word was removed in version 1.1.0. The arithmetic does not establish it: if the holder of the signing key rewrote history and re-signed it, every check still passes. What the result claims now depends on where the key came from. Where the key was taken from the bundle, the tool states that the file is self-consistent and that whoever assembled it held the matching private key — and says plainly that this does not establish who that was, nor detect a rewrite re-signed by that holder. **Where the key was supplied independently, the tool makes the stronger claim without hedging: the covered entries are unchanged relative to a commitment obtained independently of the file.**

9.4 What a passing result does not prove

Four things, each stated in the tool's own output rather than only here.

It does not prove the record is complete. A bundle truncated at an authentic earlier checkpoint is internally perfect: every record genuine, every link recomputing, the signature valid. **A hash chain cannot establish that the record continues beyond the point shown.** The tool therefore states the date and coverage of the most recent signature *in the file* and says plainly that whether the record continues beyond it cannot be determined from the file. A recipient must obtain the latest checkpoint's date and coverage through a channel independent of the bundle — the same channel required for the public key.

Two things bound the exposure. The bundle carries its own summary of what it should contain, and the tool cross-checks four fields including the recomputed tip; **a party who truncates without also making the summary self-consistent produces a file that contradicts itself**, which the tool reports as an inconsistency for the recipient to resolve with the sender — not as evidence of wrongdoing, since the summary is unsigned and proves nothing on its own. A party who does make it consistent defeats that check, and only the freshness caveat remains.

This was a defect in the published tool until 9 August 2026. Before that correction, a truncated bundle produced the verdict *"the record is intact; nothing has been added, removed, altered or re-ordered"* while records had in fact been removed. §11 records it.

It does not prove the withheld records are innocuous. The assembler chooses what to disclose. The tool reports the count and proportion withheld and states that a withheld record is unexamined rather than verified. **Withholding permits concealment, not falsification** — the published fingerprint remains a binding commitment, so a different version cannot later be substituted.

It does not prove the verdict followed the policy. §7.7. The tool verifies the integrity of what was recorded, not the correctness of what was decided.

It does not prove the record is true. The chain establishes what was declared and decided, not that the declaration described what happened.

9.5 The trust model, and the honest weakness

The key matters more than the tool. A bundle that carries its own public key proves only that whoever built it held the corresponding private key — not who that was. The tool says this in its output whenever the key came from the bundle, and switches to a stronger statement when the key was supplied on the command line: *this result binds the record to a key you obtained independently*.

But the independent channel is still OQIRON's. The key is published at `oqiron.ai/verify`, which OQIRON controls. This defends against substitution of an evidence file; it does not defend against OQIRON itself, since a party controlling both the evidence and the publication channel could present a rewritten history with a matching key. **§7.4 and §1.2 state the position: operator-independent verification requires trust material retained by a party other than OQIRON.** The tool's design supports that — it accepts any key on the command line — but the practice of external retention is not established. §11 records it.

9.6 The published limits

The tool's own documentation names five limits in the same register as this document, so that a reader who downloads it and never reads this paper still finds them: concealment by withholding; a removed signature enlarging the unsigned tail; a stale record presented as current; that unchanged is not the same as true; and that gaps in the sequence numbering are normal rather than evidence of removal.

The tool's failure paths have been proven by inducing them, not by observing their absence. Sixteen attack cases were run against deliberately corrupted bundles: rewritten verdicts, recomputed hashes, a fully re-chained forgery, deletion, insertion, reordering, an altered genesis value, forged and borrowed signatures, altered commitments on withheld records, duplicate sequence numbers, and malformed files. Thirteen were caught with distinct exit codes and stated reasons. Three were not, and each is documented above or in §11: reordering the file's array without changing sequence numbers is meaningless by design; removing the most recent checkpoint produces a large unsigned tail rather than a failure, which the tool reports prominently; and downgrading a disclosed record to withheld is the concealment case.

The result model distinguishes four states, in the structured output and in the exit status: verified, invalid, inconclusive, and malformed. A self-contradicting bundle returns *inconclusive* — the earlier reasoning, that a non-zero exit asserts a failure the tool cannot substantiate, was a false binary, since the tool can decline to conclude. Inconclusive fires on self-contradiction only: an uncovered tail, an old signature, and a key taken from the bundle all remain verified with the condition reported in prose, because making any of them inconclusive would fire on nearly every legitimate bundle and drain the state of meaning.

The exit contract changed in version 1.1.0 and a consumer branching on the numeric code from 1.0.0 must be updated; one treating any non-zero result as "do not rely on this" is correct under both. §11 records it.

9.7 What a recipient should do

Stated as a procedure, because a verification tool that is used incorrectly proves nothing:

1. Obtain the tool from the public repository rather than from the party presenting the record. **A verifier supplied by the party under examination is not independent** — the same reasoning as §5.6's note that a signed pin does not cover the program that checks it.
2. Obtain the public key from `oqiron.ai/verify`, or better, from a copy retained before the period in question.
3. Run the tool with the key supplied explicitly, not the one carried in the bundle.
4. Read the count of withheld records and treat them as unexamined.
5. Confirm the date and coverage of the latest checkpoint from a source other than the bundle.
6. Retain the checkpoint. **A checkpoint retained today is what makes verification operator-independent tomorrow** — this is the single most useful thing a recipient can do, and it costs nothing.

Step 6 is the one most likely to be skipped and the one that matters most.

10. Platform and deployment

As built throughout, except where marked. This section describes what the rail actually runs on, what protects it, and where the hardening is incomplete. Figures are as at the date on the cover.

10.1 Topology

A single host runs the whole rail. Traffic terminates TLS at a reverse proxy, which forwards to an application server over the loopback interface; the application talks to a PostgreSQL instance also bound to loopback.

```
Internet ──TLS──► reverse proxy ──► application server ──► PostgreSQL
                  (:80, :443)        (loopback :5001)       (loopback :5432)
```

Externally reachable: the two web ports and SSH. Everything else — the application, the database, an in-memory cache, and five sibling applications on the same host — is bound to loopback and unreachable from outside.

Certificates are issued by a public certificate authority and renewed automatically.

One thing on this host is not MURAQIB. A separate application belonging to a different system listens on two external ports. It shares the host and nothing else: no database, no credentials, no code path. It is named here because a reader scanning the host will find it, and because host-sharing is a real consideration for a deployment that claims isolation. A dedicated or on-premise deployment does not have this property.

The application server runs four synchronous workers, each recycled after a bounded number of requests. Two consequences follow and are stated because they surprise people: **the rail handles four concurrent clearance requests**, and **a change written to the application directory becomes live within roughly an hour without any restart**, as workers recycle. The second matters operationally — there is no deployment gate between writing a file and running it.

The database connection pool holds twenty connections with ten in overflow, and fails closed on saturation rather than queuing indefinitely. A saturated pool returns the non-verdict described in §6.

10.2 What is not in place

Stated before the rest, because it bounds everything else in this section.

There is no redundancy. One region, one host, one instance. No active-active, no failover, no standby. Combined with the fail-closed contract (§3.6), an outage of this host is an outage of every governed action.

There is no degraded mode and no local decision cache. A caller that cannot reach the rail cannot proceed, by design.

There is no defined recovery posture — no stated recovery time objective, no stated recovery point objective, and no tested failover procedure. Backups exist and restoration has been exercised; a full disaster recovery has not been designed.

For a system this document describes as availability-critical infrastructure, that gap is material. §11 records it.

10.3 Credentials and sessions

Agent credentials are API keys, validated against a stored SHA-256 digest. The digest is unsalted and uses no key-derivation function. For randomly generated machine credentials of sufficient length this is materially different from password storage and is not exposed to dictionary attack — the argument depends entirely on generation entropy, which is why §11 records both the storage choice and the obligation to state the generator's entropy so the argument is testable rather than asserted. It remains weaker than a keyed construction.

Keys carry scopes. An API key lacking the scope required for an endpoint is refused distinctly from one that fails authentication. **That check is not reached when the caller presents a bearer token, and the token endpoint issues one to any valid key without reading its scopes**, so a scope list on a key record describes how it was issued and not what it can do. Recorded and demonstrated at 11.2.23.

Keys are accepted in a request header only. A query-parameter fallback was removed on 9 August 2026; a URL carrying a credential is retained in proxy logs, their archives, and backups. No occurrence was found in the retained log window.

Human credentials are stored with bcrypt at cost 12, applied over a normalising pre-hash so that no password is truncated by bcrypt's input-length limit. The pre-hash is a SHA-256 digest rendered as hexadecimal, which produces a fixed-length ASCII input containing no null bytes. **This construction should be stated precisely rather than described as "a pre-hash", because pre-hashing before bcrypt is a pattern with known failure modes** — a raw binary digest can contain a null byte and truncate the input, and a scheme that hashes only the password without domain separation can be susceptible to credential-shucking where the same digest is used elsewhere. Neither applies here; the digest is hexadecimal and is used for no other purpose. A newer memory-hard function would be the current recommendation for a new system, and §11 records the choice.

Sessions are signed cookies marked secure, HTTP-only, and same-site lax. **That same-site policy is the entirety of the cross-site protection on the clearance endpoint**: there is no CSRF token and no origin or referer check. It withholds the cookie on a cross-site form submission, which defeats the plain attack, but it does not cover a same-site context such as a subdomain. §11 records it.

Host access is by SSH key only, with password authentication disabled and repeated-failure blocking in place.

10.4 Rate limiting

Rate limits are applied at the reverse proxy to authentication, token exchange, and unauthenticated reads.

Until 10 August 2026 there was no rate limit on the clearance endpoint. A per-credential rate and concurrency limit is now in force: requests are keyed on the credential presented rather than on the source address, so a single flooding caller cannot starve others sharing its address, and a concurrency ceiling leaves at least half the rail available to everyone else. The rate was sized from measured traffic — roughly six times the busiest genuine production caller.

The underlying capacity is unchanged, and it is the real constraint. Four synchronous workers serve one request each; the connection pool, at twenty per worker with ten in overflow, is nowhere near binding. **Workers are what saturate.** Beyond four in flight, requests queue in the listen backlog until a timeout fires. No fair-admission scheme distinguishes tenants or reserves capacity, and the instance is burstable, so sustained load above its baseline throttles it below even that figure. §11 records the residual.

Request size is bounded: a declared summary above 16,384 characters is refused naming the limit, a request body above one mebibyte is refused before the handler runs, and the proxy refuses anything above its own larger limit. The summary bound exists because the personal-data scan is unbounded in cost with respect to input length; under the cap, worst-case scan cost is measured under nine milliseconds against deliberately adversarial input.

10.5 Tenancy

Tenant attribution is derived server-side and sealed into the evidence record. Tenant isolation is not enforced.

The distinction is important and the current state should not be read as partial isolation. Attribution means a record states which tenant an action belonged to. Isolation would mean a principal can read only its own tenant's records. **The second is not in force**: no query on any evidence-returning route carries a tenant predicate today, and a principal with access to those routes can read every tenant's records — metadata, sealed payloads including authoriser identities, and checkpoints.

What bounds this today is that no external party holds a credential. Every principal able to reach those routes is operated by OQIRON, and the deployment has one active human account. **That is a property of the current deployment, not of the architecture**, and it ceases the moment a customer credential is issued.

The machinery to enforce isolation exists and is proven: predicates, scope resolution, and grant provisioning are built and tested, and six routes returning full payloads are wired. It is not switched on, because switching it on today would scope six routes while leaving twelve unscoped — which would report isolation while not providing it, and that is worse than the current honest state. Completing the remaining routes is next-phase work.

Two properties of the design are worth stating because they will matter to a reader assessing it. A scoped response reports **how many records were withheld and why**, so a tenant view cannot silently omit two-thirds of the record. And **a principal with no grant sees nothing and is told that this is a provisioning gap rather than an empty dataset** — a silent empty result would be the more dangerous failure.

Sixty-two percent of the chain predates tenant attribution and cannot be attributed retroactively, because the records that would carry the attribution are sealed. Those records are excluded from a tenant-scoped view and counted in what the view reports as withheld.

§11 records the isolation gap as the most significant item in this section.

10.6 Backups and monitoring

Backups are taken nightly, encrypted, and pushed to a separate host. The encryption key is held by the operator and is not present on either machine, so the receiving host cannot read what it stores. Restoration has been exercised.

That arrangement is deliberately one-way and it has a consequence stated in §7.5: because nothing on the production host can decrypt the copy, the copy cannot serve as corroboration during an anchoring ceremony. It protects against loss and against an attacker who reaches the production host; it does not protect against an attacker whose manipulation was faithfully copied.

Monitoring covers host reachability, service health, and the age of the evidence chain's most recent countersignature. The anchor watcher stages a signable digest and alerts when the chain has gone unanchored; it cannot anchor by itself, because the signing key is deliberately offline. **The loop therefore closes only when a person acts.** §7.6 states the consequence for the tamper window.

Two operational defects are recorded rather than smoothed over. The anchor watcher is **scheduled twice by two independent mechanisms**, sending alerts to two different destinations; the staging step is idempotent so no harm results, but it is clutter and it splits the alerting. And **three of five behavioural anomaly detectors have never produced a result**: a query comparing a text column to a timestamp without a cast fails, poisons the surrounding transaction, and prevents the two detectors after it from running at all — with every failure discarded silently. Rubber-stamp approval detection, financial-outlier detection and off-hours activity detection have therefore never run, while the interface reported that no anomalies were found.

That last defect is the reason the interface's wording was corrected on 9 August 2026 to say that no anomalies were found *in the records examined* rather than asserting that all agents are within normal parameters. **The detectors themselves are not yet fixed**, and a corrected off-hours detector would fire today on at least one agent. §11 records both.

10.7 Capacity

No throughput measurement is claimed. Figures from earlier load testing predate the transaction and locking changes described in §6 and no longer describe the running system; re-measurement has not been performed. Stating a stale number would be worse than stating none.

What can be stated is the architectural bound: **four synchronous workers, so four concurrent clearance requests**, with a connection pool of twenty plus ten that fails closed beyond that.

One consequence of the §6 correction deserves naming. Chain appends are now serialised by an advisory lock that genuinely holds — previously it was released immediately by a connection-mode defect, so appends ran in parallel and collided. **Serialisation is correct for a hash chain**, since entries must be

totally ordered, but it means sustained concurrent clearance now queues on the append rather than racing, and a single attempt that cannot acquire in time fails closed rather than retrying. The throughput characteristic of the corrected system is therefore different from the uncorrected one, and neither has been measured. §11 records the measurement obligation.

For a rail intended to sit synchronously in front of every consequential action, latency and throughput are architectural properties rather than performance footnotes, and this section cannot currently state them.

10.8 On-premise deployment

Objective. The clearance contract is identical on-premise and hosted (R3). What differs is everything an institution must supply itself: TLS certificates and their renewal, the reverse proxy and its rate limits, a PostgreSQL instance, process supervision, backup and restoration, monitoring, and — most consequentially — **custody of the offline anchor key and the discipline of the anchoring ceremony**.

That last item is not a deployment detail. Every property in §7 that depends on the signing key being outside the production environment depends, on-premise, on the customer's own key custody. **The reasoning in §3.4 about a single-custodian scheme applies equally to the customer's custodian**, and the mitigation is the same: a checkpoint retained by a party other than the custodian.

An on-premise deployment does not currently exist. The claim in this section is architectural, not observed.

11. Limitations

As built throughout, except where a remedy is marked **Objective**.

This section collects every limitation stated anywhere in this document into one place, so that a reader who reads nothing else can read this. Nothing appears here that is not also stated in the section it belongs to, and nothing stated elsewhere is omitted here. If you find a limitation in the body that is missing from this register, that is a defect and we would like to know.

11.0 How this section is organised

A list of a hundred items is not a disclosure; it is a place to hide one. The items below are grouped by **what they should change about a reader's decision**, and within that by whether they are:

Inherent boundary — a limit of what any system of this kind can establish. These do not go away with engineering effort, funding, or time. They tell a reader what to expect forever.

Current architectural limitation — a property of the architecture as it stands today, which could be engineered away by changing the design. These are real constraints now and are not permanent.

Deployment immaturity — implementation and operational gaps that are not architectural boundaries at all. They reflect the stage of the programme rather than the shape of the design.

The three carry different weight, and conflating them is how a reader is misled in either direction. We have avoided attaching a timescale to the third category: some items in it are a week's work and some are a year's.

11.1 What would stop a deployment today

Six items. A reader assessing MURAQIB for production use should resolve these before anything else.

11.1.1 There is no enforcement point

Current architectural limitation. §2.1. MURAQIB returns a verdict; nothing prevents the action. A caller that receives BLOCKED and proceeds is not stopped, and a caller that never asks is not observed. Detection depends on the institution correlating effects against clearances, which OQIRON does not ship a tool for, and some actions — data egress in particular — leave no correlatable effect at all. **Objective:** resource-side enforcement. Not started; no component exists.

11.1.2 Tenant read isolation is not enforced

Deployment immaturity. §10.5. No query on any evidence-returning route carries a tenant predicate. A principal with access to those routes reads every tenant's records — metadata, sealed payloads including authoriser identities, and checkpoints. What bounds this today is that no external party holds a credential; that is a property of the current deployment, not of the architecture. **The consequence for a buyer: a**

hosted deployment must not issue an external credential until isolation is complete and adversarially tested. The machinery is built and proven; six routes returning payloads are wired; twelve remain. Completing them is next-phase work, and until it is done no external credential can be issued.

11.1.3 The rail is a single failure domain, and callers fail closed

Deployment immaturity. §§3.6, 10.2. One region, one host, one instance, four synchronous workers. No active-active, no failover, no standby, no degraded mode, no local cache. No stated recovery time or recovery point objective and no tested disaster recovery. Combined with the fail-closed contract, an outage of this host is an outage of every governed action.

11.1.4 The evidence guarantee rests on a single custodian with no key lifecycle

Inherent boundary in part, deployment immaturity in part. §§3.4, 3.8, 7.4. Cryptography defeats a compromised server; it does not defeat the key holder. That is inherent to single-custodian signing. What is not inherent, and is simply not built: there is no key rotation procedure, no revocation mechanism, and no defined position on the evidentiary standing of records anchored before a compromise is discovered. For infrastructure whose evidentiary life is measured in years, this is a prerequisite rather than a refinement. **The available mitigation is external retention of checkpoints, and it is not systematised.** §11.2.1.

11.1.5 A decision cannot be independently replayed

Current architectural limitation; Objective to close. §7.7. Of the twelve server-derived conditions the controls read, two are sealed. A record establishes what was decided, on what declaration, by which catalogue — not that the verdict was the correct result of applying that catalogue to those inputs. **MURAQIB has tamper-evident logging; it does not yet have verifiable policy execution.** Closing this is the headline capability of the next phase.

11.1.6 Live credentials remain exposed in historical backups

Deployment immaturity, open. 11.5.6. Twenty-three plaintext backup bundles on the production host contain currently-valid credentials: the session signing key, the token signing secret, the database password, five agent API credentials and fifteen TLS private keys. Encrypting future backups does not invalidate credentials already copied. **The exposure ends when those credentials are rotated, not when the backup format changes**, and no rotation procedure exists for any of them — 11.5.5. They are retained only until the new backup arrangement is confirmed end to end, after which they should be securely destroyed.

11.2 What a recipient of evidence must understand

11.2.1 Operator-independent verification is conditional

Inherent boundary. §§1.2, 3.4, 7.4, 9.5. R4 holds only where the verifier possesses verification material obtained independently of the party presenting the record. The public key is published on a channel separate from any evidence file, but that channel is OQIRON's; a party controlling both the evidence and

the publication channel could present a rewritten history with a matching key. **A recipient who retains nothing is trusting OQIRON.** There is no scheduled distribution of checkpoints to independent holders, no external witness, and no third-party timestamping.

11.2.2 A chain cannot prove it has been shown in full

Inherent boundary. §§3.7, 7.2, 9.4. A record truncated at an authentic earlier checkpoint is internally perfect. The tool cross-checks the bundle's own summary against its contents, so a careless truncation contradicts itself and is reported — but a consistent one is undetectable from the file. The remedy is external: confirm the latest checkpoint's date and coverage through an independent channel.

11.2.3 Withholding permits concealment

Inherent boundary. §§3.7, 9.4. The party assembling a bundle chooses what to disclose, and the verifier cannot distinguish a record withheld for confidentiality from one withheld because it is unfavourable. It reports the count and states that a withheld record is unexamined. **Withholding cannot permit falsification** — the published fingerprint remains a binding commitment for records covered by a trusted checkpoint.

11.2.4 The record does not establish truth

Inherent boundary. §1.6. The chain establishes what was declared and decided, not that the declaration described the action performed.

11.2.5 The tamper window depends on manual discipline

Deployment immaturity. §7.6. Anchoring is operator-driven, not scheduled; intervals have run to nearly four days and 135 events. No maximum elapsed time, no maximum unanchored count, no alert threshold, and no defined posture when exceeded. The verifier's ten-percent warning is a presentational aid, not a security boundary.

11.2.6 The signing ceremony authenticates the database's claim about itself

Deployment immaturity. §7.5. The signer is presented with a sequence number and a digest computed from the production database. The three insertion guards re-read the same source and establish self-consistency rather than authenticity. **An adversary who rewrites the unsigned tail and waits for the next routine ceremony may have the manipulation legitimately countersigned.** The off-host backup cannot corroborate, being encrypted to a key the production host does not hold.

11.2.7 Some evidence is absent by construction

Inherent boundary. §§5.6, 6.4, 7.7. Requests refused before evaluation — bad credential, unresolved environment, over-length input — write nothing to the chain. Authentication failures leave no trace in the evidence record. Policy history begins 8 August 2026; the 433 decisions sealed before it carry no catalogue digest and their preceding policy changes cannot be added.

11.2.9 One key signs two things, with no domain separation

Current architectural limitation. §7.4. The same offline key countersigns the evidence chain tip and the policy catalogue digest, and in both cases signs a bare hexadecimal digest with no purpose prefix or context string. A signature over one digest does not verify against a different digest; the weakness is that **the two protocols occupy the same signed-message namespace**, so a signed message carries no cryptographic marker of which purpose it was produced for. The remedy is separate keys and a domain-separating prefix. Because every existing signature was made without one, correcting it means either accepting two formats indefinitely or introducing the change with a key rotation, so it is scheduled with the key-lifecycle work in 11.1.4 rather than patched.

11.2.10 Privileged database control can defeat storage-layer immutability

Inherent boundary of a database-enforced control. §§3.3, 7.3. Append-only enforcement rests on a trigger and on uniqueness constraints. While those remain in force, no application-level race can persist a fork and no application compromise can alter a record. **An operator with schema-level privileges can remove them and then modify or delete rows freely.** What they cannot do is produce a countersignature over the result. Destruction is the residue: a superuser can delete evidence, and nothing recovers it except backups — the chain detects modification, it does not confer durability.

11.2.11 The verifier's exit status does not express freshness

Deployment immaturity — partly closed. §9.6. Version 1.0.0 of the published tool exited successfully on a self-contradicting bundle, so an automated consumer checking only the exit status learned nothing about completeness. Version 1.1.0 introduces an explicit inconclusive result and a distinct exit code, and **the exit contract therefore changed**: a consumer branching on the numeric code from 1.0.0 must be updated. A carefully truncated bundle whose summary is made self-consistent still returns a successful result, because the file contains no evidence of truncation; 11.2.2 states why no change inside the tool can close that.

11.2.12 The verifier's own provenance must be established independently

Inherent boundary. §9.7. A verifier obtained from the party under examination is not independent, and its self-tests do not change that: an adversary able to modify the tool can modify its self-tests and their expected outputs. Those tests protect against accidental corruption and implementation regression, not against a supplied verifier. A recipient must obtain the tool independently and, ideally, pin a release hash retained before the period in question.

11.2.13 The canonical form is not RFC 8785

Current architectural limitation, chosen. §7.1. The serialisation is fully specified and independently reproduced, but it is not the JSON Canonicalization Scheme of RFC 8785. Numbers and booleans are coerced to strings before serialisation, so the canonical bytes carry `"7"` and `"true"` where JCS emits `7` and `true`; key ordering follows Unicode code point rather than the UTF-16 code-unit order JCS mandates, which is unobservable for the ASCII field names in use and is still a formal divergence. **The choice was deliberate and is not reversible here**: string payloads prevent the database's own

numeric normalisation from altering a value on re-read, and the format is now bound into every sealed record, so changing it would invalidate every existing hash. An implementer must follow the published specification rather than assume a standard scheme; adopting one is a question for a future chain lineage.

11.2.14 Timestamps are not normalised before hashing

Current architectural limitation. §7.1, B.6(b). The canonical encoder applies no date handling: `created_at` is hashed as the string the caller supplied, and two records denoting the same instant in different formats produce different hashes. Callers must pre-format timestamps; nothing in the rail enforces a format, and nothing detects a caller that changes format between records. **This does not affect chain integrity** — a record verifies against whatever string was sealed — but a party comparing records across callers cannot assume timestamp comparability from hash agreement.

11.2.15 Stored payload text is not the hashed bytes

Current architectural limitation. §7.1, B.6(c). The `payload_json` column is written with a different serialisation from the one that is hashed: the hash is computed over canonical bytes with sorted keys and no whitespace, while the stored text carries neither guarantee. **A verifier must re-canonicalise from the parsed structure; hashing the stored text directly will not reproduce the content hash.** The published verifier does this correctly, and the point is recorded here because an independent reimplementation that skips it will report a sound chain as broken.

11.2.16 A valid hash does not imply a non-empty record

Deployment immaturity. §7.1, B.6(d). The writer asserts no minimum content: a record with an empty payload seals and verifies cleanly, and one exists in the sandbox lineage. Hash validity attests that a record has not been altered since it was written, not that it says anything. **An examiner must read the payload, not merely confirm the seal.** No production record is affected.

11.2.17 The event type is not covered by the hash

Current architectural limitation. §7.1, B.6(a). Each chain entry carries an `event_type` — `EVIDENCE_CREATED`, `POLICY_CHANGED`, and the privilege events — stored as its own column beside the payload rather than inside it. The content hash is computed over the payload alone, so **a matching hash attests to what a record contains, not to how it is classified.** A verifier confirming that a chain is intact has not confirmed that each entry is the kind of entry it says it is. No misclassification is known; the property is recorded because a reader counting event types from a verified bundle is relying on a field the verification did not cover.

11.2.18 Approval provenance is unsealed and mutable

Current architectural limitation. §§4.10, 7.7, B.2. The post-write-mutable fields — `approved_by`, `approved_at`, `approval_note` and `pushed_to_majlis` — are excluded from the hashed payload by the allow-list, which is correct and deliberate: a decision's content hash attests to the decision as recorded at write time, and a later human act belongs in a later event rather than retrospectively inside an earlier one. **A human resolution IS sealed, as a `DECISION_RESOLVED` event** carrying the outcome, the resolver's session-derived identity and role, the time, and the resolution note (§11.2.18). The columns live in `evidence_log`, which is not one of the three append-only tables, carries no trigger, and on which the

application role holds `UPDATE`, so **the column values can be added, altered or removed in place without breaking any seal and without a superuser.** That is the reason they are **not the record**. The record is the sealed event, the resolution state of every sealed escalation is recoverable from the chain alone, and altering the columns cannot alter what the chain says. `pushed_to_majlis` has no sealing event and remains unsealed.

Two production records carry an approval — `MRQ-2026-000001` and `MRQ-2026-000005`, both from May 2026 — and neither has a chain event of any type, having been written before the chain existed. **A content hash attests to the decision as it was recorded at write time. It attests to nothing about who approved that decision, when, or whether an approval recorded today was there yesterday.** Sealing approval events is not designed.

11.2.19 There is no self-service path to an evidence bundle

Current architectural limitation. §§1.2, 3.7, 9.2. A third party holding an evidence bundle can verify it without OQIRON's cooperation. Obtaining the bundle requires OQIRON. Bundle generation is a command-line operation performed by an operator on the host; no HTTP route emits a bundle, authenticated or otherwise, and the two export routes that do exist produce spreadsheet and document extracts rather than a verifiable bundle.

The practical consequence is that a Controller, or an assessor appointed on its behalf, receives evidence on a schedule OQIRON controls. Verification is operator-independent; acquisition is not.

This is distinct from 11.2.3, which concerns what an assembler chooses to include in a bundle it has decided to produce. This item concerns whether one can be produced at all without OQIRON's participation.

11.2.26 Credential revocation

Resolved 12–13 September 2026. §§4.2, 7.3, 11.2.23. This item previously recorded that **no production credential could be revoked by any code in this system**, that revocation was agent-granular, and that withdrawing a credential required a database session on the host. That was accurate when measured on 11 September 2026 and is no longer the state of the system. It is retained rather than deleted because the four revocations of 11 September were performed under the condition it describes.

One mechanism, two doors. A single function performs every revocation. A host-side command-line tool and an authenticated HTTP route both call it; neither carries its own implementation. The command-line path needs no session and no working application, which is the point of keeping it: it works when the application is the thing that is broken. The route is what an operator reaches from a phone.

Revocation is key-granular. One credential is withdrawn without disturbing any other credential the same agent holds. The target is named by a hash prefix of at least twelve hexadecimal characters, never by the credential itself, and an ambiguous prefix is refused rather than resolved. A reason of at least eight characters is mandatory and is recorded. Taking an agent's last live credential requires a second, explicit confirmation that names the consequence.

The order is database, then file, then seal, and the order is load-bearing. The database alone decides whether a credential authenticates, so it is closed first; a failure to mirror the change into the secondary store leaves a credential that is already dead. The reverse order would leave a credential that still authenticates while every store reports it revoked. The seal is last and **cannot gate the revocation**: if sealing fails the credential stays revoked and the failure is reported loudly, because a credential dying matters more than the record of it dying.

Tokens already issued are reached. A token now carries a key identifier and is validated against the state of the key that minted it, so a token bearing a revoked credential stops authenticating on its next call rather than surviving to the end of its twenty-four-hour life.

Demonstrated on production credentials. On 13 September 2026 the eight production credentials unused since July were revoked through this path, one at a time, each with its own captured prior state and its own sealed event at chain sequences 489 to 496. Each was verified against the authenticating code path after the write. One production credential remains in force.

What this item does not resolve. There is still **no expiry field** on the key record, so nothing lapses on its own and an unrevoked credential remains valid indefinitely. Issuance remains unsealed (11.2.25). And a revocation withdraws a credential without deregistering the agent, which is the correct separation but means an inventory of registered agents will not show it.

11.2.25 Credential issuance is recorded nowhere

Current architectural limitation, halved 12–13 September 2026. §§5.1, 7.3, 11.2.18. Measured 11 September 2026, when neither issuing a credential nor revoking one emitted a chain event: across 443 sealed events there were five event types and none concerned a credential.

The revocation half is closed. `CREDENTIAL_REVOKED` is now a sealed event type carrying the key identifier, the agent, the scopes withdrawn, the operator and the mandatory reason. Thirteen such events stand in the production chain as of 13 September 2026, which now holds 458 sealed events across six event types. **Issuance remains unsealed**, so the record is asymmetric: the chain shows every credential withdrawn and no credential granted.

The remaining absence is an omission rather than a limit of the chain. The chain already records a privilege change of a different kind — `PRIVILEGE_NARROWED` and `PRIVILEGE_RESTORED` are sealed event types, and both have been emitted — and now records credential withdrawal by the same mechanism. Applying it to issuance is outstanding work, not a constraint.

A successful issuance writes no log line either. The credential logger emits on failure paths only: a registry that could not be read, a refusal for an unknown agent, a generation that raised, and a store that could not be written. The only issuance events recorded anywhere in the system are therefore **the ones that did not issue a credential**. A mint that succeeds is silent.

What remains as evidence of an issuance is a single mutable column — the creation timestamp on the key record — carrying no actor. The owner field beside it records a claimed owner taken from the issuing session, not an authenticated minter, and neither field is covered by any hash.

Revocation formerly left still less: the active flag moved from true to false in place, with no timestamp, no actor and no event, and the 11 September revocations are evidenced only by operator-written capture files held outside the chain. From 12 September a revocation seals an event naming the credential, the operator and the reason, so a reader can establish what was withdrawn and why without relying on a record OQIRON produced about itself. The key record itself still carries no revocation timestamp and no actor column; the evidence is in the chain, not in the store.

11.2.24 Every route is internet-reachable; protection is per-route, not positional

Current architectural limitation. §§7.1, 9.2, 11.2.19. The reverse proxy forwards `location /` to the application in its entirety. There is no route allowlist, no network segmentation, and no administrative interface bound to a separate address or port. Every route registered in the application becomes reachable from the public internet on deployment, and each is protected only by the checks written inside its own handler.

Measured 31 August 2026: of thirty-nine parameter-free `GET` routes, four respond without authentication — `/`, `/login`, `/auth/logout` and `/health` — and every other route, including every administrative and state-changing route, returns 401 or 403. The posture is therefore correct today. What it is not is structural: a route added without its decorator is exposed at the moment it is deployed, and nothing outside the handler would prevent that. A reviewer should read the authentication of this system as a property of thirty-nine independent handlers rather than of its network position.

11.2.23 A credential's scopes do not bind it

Current architectural limitation. §§4.2, 7.3. Measured 11 September 2026; **materially corrected 13 September 2026.** A credential's authority is carried by a `scopes` array on the key record. This item previously stated that the array is enforced fail-closed at both clearance routes. That was wrong, and the correction is larger than the error.

No scope restricts what a credential can do. Every credential on this system can clear actions, whatever its recorded scopes, and can do so without acquiring any privilege it did not already hold. There is no narrow credential. A credential recorded as holding `read` alone has, in effect, the authority of one holding everything, and the record that says otherwise is the only thing standing between a reader and that conclusion.

The mechanism, stated so the claim can be checked rather than taken. Exactly two scope strings are ever compared against anything anywhere in the system. `intercept` is tested at the two clearance routes, and `read` at the evidence-chain verification route. A third scope, `write`, is held by key records and tested nowhere. The `intercept` test sits in the API-key branch of each clearance route, below a bearer-token branch that is tried first and that never reads a token's scopes; the token endpoint issues a twenty-four-hour token to any valid credential without reading its scopes either. **A holder therefore steps past the only test that guards acting, by exchanging its own credential for a token.** The single scope test that cannot be sidestepped is `read` at chain verification, where both doors are checked — and that route grants reading the evidence, not acting.

A reader who tests this will find the refusal first. Presenting a read-only credential as an API key returns a scope refusal, which is what this item previously reported and what any single test will show. That refusal is not enforcement; it is the bypassable half of it. Enforcement present on the door an assessor tests and absent on the door an integrator uses is worse than none, because the test returns a result that reads as proof. The one production integration in service authenticates by token and has therefore never reached the scope test at all.

Induced 13 September 2026 on a read-only credential minted for the purpose. Presented as an API key to the clearance route, it was refused with `insufficient_scope`, exactly as this item previously implied it must be. The same credential was then exchanged for a token, which was issued and carried `scopes: ["read"]`, and that token cleared a 250,000 SAR vendor payment at the same route: accepted, escalated, and sealed into the chain. Two calls. The payload-scanning route behaves identically. The proof credential was revoked and the revocation sealed.

No interface can express a scope. `generate_api_key` accepts a `scopes` argument and honours it, but neither path that issues a credential supplies one that differs from the default. The HTTP route reads only an agent identity, an agent name and an owner from the request body; the host-side sandbox tool passes the default pair explicitly. Every credential mintable today therefore carries `intercept, read`, and a credential that needs only clearance cannot be issued without also granting read.

The store is not uniform, and the non-uniformity is decorative. Forty key records carry four distinct scope sets. Until 13 September 2026 twelve credentials were active, of which eight carried fewer scopes than the issuance default and one carried a third scope, `write`, that no code in the system reads. Eight were revoked on 13 September, leaving **four active: three sandbox credentials holding `intercept, read` and one production credential holding `intercept`**. Re-issuing any narrow credential **widens** it, silently, to the default pair, so the narrow ones in force cannot be reproduced through any interface. Given the enforcement gap above, that widening changes what the inventory says rather than what the credential can do, which is the more dangerous of the two.

A key does not carry its environment. The three sandbox credentials in force hold `intercept, read`, a broader set than the single production credential that holds `intercept` alone. Separation between sandbox and production is established by the agent's registry row (11.2.22), never by the key, and scope must not be read as a proxy for it.

Recorded in the same measurement: the foreign key from the key store to the agent registry is declared `NOT VALID`, so rows predating it were never checked. Four credentials reached that state, referencing agent identities with no registry row. They were revoked on 11 September 2026, and no active credential now references an absent agent. **The constraint that appears to forbid the condition still does not enforce it**, and validating it remains outstanding.

11.2.22 Environment routing rests on an unconstrained column, and the naming convention is not a control

Current architectural limitation. §§4.9, 7.7. Whether an agent's sealed records enter the anchored production chain or the sandbox chain is decided solely by one column on the agent registry: `text`, not null, with no default, no check constraint and no enumerated type. The resolver performs an equality

lookup on the agent identity and returns that column; `prod` routes to production, `dev` and `test` to sandbox, and any other value fails closed with the declaration refused before any write.

The agent's name has no bearing on this. Identities issued for sandbox use conventionally begin `SANDBOX-`, and all twelve registered agents correlate that prefix with their environment — twelve of twelve, without exception. No code anywhere reads the prefix. An agent named `SANDBOX-` with the column set to `prod` writes to the chain that is anchored and disclosed; one named otherwise with the column set to `dev` does not. The convention is a label for human readers and must not be read as a mechanism.

Two consequences follow. The word `sandbox` is not an accepted value: a row written with it fails closed and yields an identity that can never transact, despite that being the term used in the platform's own published material. And because the routing decision is a bare string, the property that a sandbox identity cannot reach the anchored chain is maintained by discipline rather than by the schema. A check constraint restricting the column to its three valid values does not exist.

11.2.21 Approval provenance exists as a column, not as a mechanism

Current architectural limitation. §§2 (R3), 4.4, 11.2.18. The four-eyes control described at §4.4 is enforced at the rail and its claim stands unchanged: the distinctness test runs server-side over `req.server`, and the caller's claims are never consulted for enforcement. What does not exist is any path by which a human resolves an escalated action after the fact. That gap is now partly closed: a human resolution is sealed as a `DECISION_RESOLVED` event carrying the resolver's session-derived identity, and the first such event was sealed at chain sequence 481. What remains is the submitter side. Every resolution carries `submitter_established` as an explicit value, and on every event sealed to date that value is `false`: `actor_id` is supplied by the calling system, is not a member of CANONICAL_FIELDS_V2 or V3, and is therefore not covered by an assertion signature even where one is present. Maker–checker is recorded as unevaluated on every resolution, because the maker is not established. `approved_by` is populated on three of 1,073 operational rows. One is the genuine resolution sealed at sequence 481 and derivable from the chain. The other two are from May 2026, carry no chain record of any kind, and name individuals in free text who are absent from the user store; they were written by a superseded SQLite module that has no caller and writes to a database this system no longer uses. **The derived state agrees with the columns exactly** — as at 17 September 2026, 229 unresolved and 1 resolved across the sealed escalations, with no row disagreeing — so the columns are a cache of the chain. **The cache cannot represent a rejection:** `approved_by` is written for both outcomes and only the sealed event carries `outcome`, so a rejected resolution writes the resolver's identifier into a column named `approved_by`. No rejection has been recorded yet, so this is latent; a populated `approved_by` must nonetheless not be read as evidence of approval.

This is distinct from 11.2.18, which concerns those fields being mutable and unsealed. This item concerns their having no writer at all.

11.2.20 The interface and the chain are different stores

Current architectural limitation. §§2 (R4), 4.10, B.2. A governed **decision** is written to two tables in one transaction: the sealed, hash-linked chain, and an unsealed operational store holding the decision's content. Both rows carry the same evidence identifier, and a failure to seal rolls back the operational write, so since 12 June 2026 neither exists without the other.

The twinning is a property of decisions and of nothing else, and this item must not be read as describing the chain generally. `evidence_log` is shaped for decisions — agent, action type, risk score, risk level, decision, SAAC code, tenant — and holds no row of any kind for a governance act. Of the nine event types in the chain, **seven have never had an operational-store row and never will**: credential issuance, credential revocation, agent registration, registry attestation, policy change, and privilege narrowing and restoration. Only `EVIDENCE_CREATED` is twinned; `DECISION_RESOLVED` attaches to a decision row that already exists. A reader who takes "two stores, one record" as general will conclude that an export from the mutable store is an administrative audit trail. **It contains no governance act at all.** The corollary matters equally: the unsealed legacy rows recorded below are unsealed *decisions*, and there is no parallel population of unsealed governance acts, because a governance act has only ever existed in the sealed chain or nowhere.

The limitation is what precedes that date and what reads each store. The operational store holds 1,073 rows against the chain's 453 sealed decisions; 620 correspond to no sealed decision — 563 predating the chain, 56 from a superseded identifier scheme, and one whose seal was lost to the concurrency defect. The dashboard and the spreadsheet and PDF export routes read the unsealed store, on which the application holds `UPDATE` and which carries no immutability trigger; the offline bundle reads the sealed chain. The verifier classifies the 620 as `legacy-unverifiable` and reports their count rather than including them silently.

The chain's integrity properties are unaffected. This item records that no document previously distinguished the two stores, so a reader could take a figure from the interface as a sealed figure. It is related to 11.2.18, which concerns fields the chain never covers, and distinct from it: this item concerns whole records the chain never covered.

A self-service export route is architecturally constrained rather than merely unbuilt. A bundle must contain the complete chain, genesis to tip, because verification reconstructs every link and a filtered chain cannot be verified. On a deployment serving more than one tenant, a self-service bundle would therefore disclose every other tenant's record metadata to the requester. Resolving this requires a change to the chain architecture — per-tenant chains, or inclusion proofs — not a new endpoint.

11.2.27 Agent admission is sealed; withdrawal of an agent's authority is not

Capability added and limitation exposed, 15 September 2026. §§5.0, 7.3; D5 §11 item 30. Admitting an agent to the rail now leaves a chain record — `AGENT_REGISTERED`, naming the agent, its derived environment, its permitted actions, its financial threshold and the human who admitted it. The seal is written **before** the registry row exists, so a sealing failure prevents the agent existing at all. This is the issuance ordering and it is deliberate: an unsealed live actor is the defect being closed. There is no HTTP route, by ruling — admitting an actor is a decision a named human makes — and the self-serve provisioning path calls the same command-line tool rather than writing the registry itself.

What this exposes is the other end. Freezing, suspending or revoking an agent still leaves no chain record. Four agents stand at status `revoked` with nothing naming who retired them, when, or why, one of them a production identity whose authority was withdrawn. An examiner reconstructing who could act on this rail can establish every admission from 15 September 2026 forward and **not one withdrawal, ever**.

The asymmetry is the wrong way round. Sealing the grant of authority and not its removal means the chain's account of the agent population only ever grows. Retirement needs an event type of its own, and it needs the **revocation** ordering rather than the issuance one — sealing last, and unable to be blocked by a sealing failure — because an agent losing authority matters more than the record of it losing authority. That is the same argument §11.2.26 makes for credentials, applied to the actor the credentials belong to.

11.2.28 What the chain records of platform administration

Disclosure, restated 15 September 2026. §7.3; D5 §11 item 15. The chain now carries **nine event types from eight call sites**, of which one records decisions and eight record governance acts: credential issuance and revocation, agent registration, registry attestation, policy change, privilege narrowing and restoration, and the human resolution of a held decision. At the reference deployment the governance acts total 38 events against 447 decisions.

The list of what is still unrecorded is the load-bearing half of this item. Creating a user account, changing a role, suspending or reactivating an account, amending an agent after admission, withdrawing an agent's authority, logging in or failing to, and exporting evidence all leave no chain record. **The evidence chain is a complete record of agent decisions and an incomplete record of platform administration**, and an institution should not read the second as following from the first. Where the remaining gaps are structural rather than an omission of sealing — because no route exists to seal — 11.2.27 and D5 §11 item 30 say so.

11.2.29 Published throughput is a policy limit, not a measured capacity

5 requests per second sustained, with a burst of 60, enforced per credential at the reverse proxy. Measured 5 September 2026. This is the figure a caller experiences and the figure OQIRON publishes.

It is not the rail's capacity. The rate limit binds before the application does, so the clearance path's own concurrency ceiling has never been established. Anyone reading 5 requests per second as a statement about what the system can sustain is reading it wrongly.

Nor is it the rail's rate limit. This figure describes `POST /api/v2/intercept`. It is keyed on the caller's credential — confirmed by test, and unaffected by header capitalisation. The other limited endpoints are keyed on client address, and **the majority of authenticated routes carry no rate limit at all**, being protected by authentication rather than by throttling. A reader should not generalise 5 requests per second to the rail.

The measurement has four limits, and they belong with the figure. It was generated from **one client**, whose processor saturated while producing rejected requests — the client reached its ceiling before the rail reached any. It ran against the **sandbox** chain, which takes a separate append lock from production and had **no concurrent writers**, so production under real contention could be slower. It used **one credential**, so it describes one caller's allowance rather than aggregate capacity. And the **application's ceiling was never reached**: the host's load average peaked at 0.79 while the client's processor was fully committed.

What the measurement did establish is behaviour rather than capacity. Once an agent's rolling-hour action count passed 120, control `AGT-04` escalated every subsequent request; nothing was approved after that threshold, and the system became more restrictive as load accumulated. No request was refused by the application — no `503`, no timeout, no pool exhaustion — and every rejection was a `429` from the proxy. 345 records were sealed and the chain verified intact.

11.2.8 Historical facts, disclosed

As built, and permanent.

- **One production verdict was returned without being sealed** — an escalation on 4 August 2026, under the atomicity defect corrected on 8 August. Seventy-one further instances in the sandbox lineage the same day. §6.5.
- **Two campaign authorisations were approved on unsigned four-eyes fields** — 27 June 2026, before any signing capability existed anywhere in the system. Four further submissions the same day were correctly refused by the maker-checker test. The class was brought under the signature requirement on 8 August 2026. §8.5.
- **Eleven sealed records carry a regulatory-risk flag that contributed nothing to their risk score**, owing to a field-name mismatch corrected on 8 August. Sealed records are immutable and have not been altered. §4.10.
- **A legacy route returned an approval regardless of policy, for 83 days.** Until 8 August 2026 the `/clear` endpoint applied an operating-mode transform after evaluation: where a mode file read `audit`, `monitor`, `shadow` or `advisory`, the returned decision was overwritten with APPROVED regardless of the engine's verdict. The file read `audit` continuously from 16 May to 7 August 2026, and the endpoint was internet-reachable behind a check that tested only whether an API-key header was present, not whether it was valid.

Production impact was nil, established from a record that does not depend on log retention. The path writes to a separate ledger spanning 16 February 2026 to the present; across all 495 rows, exactly one shows an approval contradicting the engine — an operator test summarised "Shadow final test", executed in `shadow` mode on 16 May 2026, where the engine's verdict of BLOCKED was itself preserved in a separate column and so was never lost. **Inside the 83-day window that ledger holds no request-written row at all**, and the retained request logs show two `/clear` requests in total, both refused with HTTP 401 before reaching the code — one of them a route-enumeration sweep from OQIRON's own coordinating host. No automated caller of that route exists on any host or in the client library.

Two points of precision. **The claim in §3.1 that an agent cannot supply a verdict is unaffected**: this was a server-side configuration transform, not an agent-supplied input. And the presence-only credential check on that route was not part of the defect and **is still present**; it is recorded at 11.5.7.

- **One record carries a tenant-lookup-failure sentinel.** §6.1.
- **Twenty-one sequence numbers are absent** from the production chain, consumed by rolled-back transactions. The chain links by hash, not by number. §7.3.
- **Version 1.0.0 of the published verification tool reported a truncated record as intact**, between 5 and 9 August 2026. Documented in ADVISORY-2026-001 in the tool's own repository. §9.4.

11.3 What the four-eyes claim does and does not establish

11.3.1 The human is not independently verified at any rung in force

Design limitation at Rungs 1 and 2; Objective at Rung 3. §8.4. Rung 2 establishes that the registered agent asserted that a named human authorised the action. It does not establish that the human did. A compromised agent holding a valid signing key can produce a correctly signed authorisation for a person who authorised nothing.

11.3.2 Distinctness is between identifiers, not persons

Inherent boundary. §8.2. The rail enforces that the asserted authoriser is not among the asserted makers. It cannot establish that two identifiers represent two people, that an identifier represents a real person, or that the asserted role belongs to them.

11.3.3 The floor is Rung 1

Deployment immaturity. §8.5. Rung 2 is enforced for five action classes. **Every other action type, including financial transfer, operates at Rung 1**, where the authoriser is whoever the calling system says it is. Six of eight production identities hold a key; the ninth active agent is a sandbox identity, and twelve are registered in total.

11.3.4 The signature does not bind the action's parameters

Current architectural limitation; Objective in the next phase. §8.3. The signed form binds the action type, the agent, the authoriser, the makers, a request reference and a timestamp. It does not bind the amount, the beneficiary, the document, or the destination. A signature authorising *a document egress* does not establish *which document, or where*.

11.3.5 There is no minimum-rung configuration, and it compounds

Deployment immaturity. §§8.5, 1.3. Assurance coverage is a property of the catalogue and changes only by changing the catalogue — which is subject to no approval or role separation. **An administrator who can edit the catalogue can lower the assurance required for an action class.**

11.3.6 Replay within an action class is not prevented

Deployment immaturity. §8.7. The signed form binds the action type and a timestamp checked against a five-minute window. The rail maintains no replay cache and does not reject a previously seen assertion. Request-reference uniqueness is a property of the caller. Staleness rejection depends on a server clock that nothing independently attests.

11.3.7 The evidence format does not state the assurance achieved

Deployment immaturity. §8.6. The signature outcome appears in three states — absent, present-but-null, and explicit — and the record does not disambiguate "not signed" from "a signature was not required here." Of 433 sealed records, 198 carry no signature field, 203 carry it as null, 27 as verified and 5 as failed.

11.3.9 The production binding format is the earlier of two

Deployment immaturity. §8.3. Two canonical binding versions are defined. **The agents signing today use the earlier form; the later version, which uses explicit field separators, is defined and not in production use.** Explicit separators exist to make field boundaries unambiguous, and their absence in the form actually in use is a question a cryptographic reviewer will raise. Migration is not scheduled, and the earlier form's byte layout has not been published with collision test vectors.

11.3.10 Agent identity proves credential possession, not attested execution

Inherent boundary as built. §3.1. Authentication establishes that a request presented a valid credential. It does not establish that the intended process holds it: **a copied API key or signing key impersonates its owner completely.** No workload attestation is built and none is on the near roadmap. This is distinct from credential rotation, which addresses exposure after the fact rather than binding a credential to an executing workload.

11.3.8 Agent signing keys have no documented lifecycle

Deployment immaturity. §8.7. Registration is by database record. A rotation has been performed but the procedure is unwritten, and there is no revocation mechanism distinct from deregistration. Key-registration changes are not sealed into the chain.

11.4 What the policy claim does and does not establish

11.4.1 Policy changes are recorded, not authorised

Deployment immaturity. §§1.3, 5.6. There is no maker-checker, no role separation, and no approval workflow over the catalogue. The declared identity and reason in a ceremony record are supplied by the person performing it. **MURAQIB cannot establish who, if anyone, approved a policy change, because it has no policy-change authorisation mechanism.**

11.4.2 The recording mechanism is inside the boundary it records

Current architectural limitation. §3.2. An administrator with write access to the application directory can disable the policy-change recorder, alter the catalogue, act, and restore both. Under honest execution of the deployed code the sequence is recorded; the recording is not an independent control against the party who can modify the code.

11.4.3 Nothing attests the deployed code

Deployment immaturity. §§3.2, 10.1. There is no code signing, no measured deployment, no immutable artifact deployment, and no separation between the authority to deploy and the authority to operate. **The policy catalogue, the application code and the service account are not separated at all**: the service runs as the account that owns the application directory, and that account can write both the catalogue it enforces and the code that enforces it. **A file written to the application directory**

becomes running code within roughly an hour, with no deployment gate. The signed catalogue pin does not cover the programs that compute and verify it, nor the evaluator that executes the catalogue. This is a straightforward hardening item and it is not done.

11.4.4 The catalogue is not the whole decision

Current architectural limitation. §5.4. Three mechanisms form a verdict: an unrecognised action type seeds an escalation before evaluation, the catalogue evaluates, and the declared-summary scan can elevate an approval. Signing the catalogue signs the largest part of the decision logic and not all of it. **The most frequently recorded determining control — 152 of 433 sealed records — is the unrecognised-action seed, which is not one of the 58.**

11.4.4b The declared-summary scanner is outside the signed artifact

Deployment immaturity. §5.4. The scan of declared fields can elevate an approval to an escalation, so it is verdict-affecting — and its rule set is not covered by the signed catalogue pin. A reader verifying the pinned catalogue has not verified everything that can change a verdict.

11.4.5 The controls do not read most declared attributes

Current architectural limitation, and an open design question. §§4.8, 5.2. Fifteen declared attributes are recorded and do not affect the verdict, including the financial amount, the data-sensitivity classification, the externality flag and the priority. A transfer of a hundred riyals and one of a hundred million receive the same treatment from the catalogue. Whether magnitude should be a policy input is open; the attributes are sealed today so that it can become one without losing history.

11.4.6 The risk score has no authority

Current architectural limitation, and an open design question. §4.10. Fourteen weighted signals produce a score and a band that no control reads, verified by exhaustive sweep. A score that cannot be evaluated is now labelled rather than dropped, and a score and its band may legitimately disagree where that occurs.

11.4.7 Change detection has limited granularity

Inherent boundary. §5.7. A change to a control's own definition names the control and the field. A change to a structure shared across controls states only that the catalogue's bytes changed.

11.4.8 Jurisdiction binding is outside the pin

Deployment immaturity. §5.7. The setting that gates two controls is a deployment-level value: not covered by the signed pin, not sealed into any record, and not logged when changed. Today only one jurisdiction is defined and every unrecognised value falls back to it, so changing it changes nothing — an accident of there being a single binding set, not a protection.

11.4.9 A legacy policy registry is exposed and is not the policy

Deployment immaturity. §5.5. The rail exposes a larger set of policy records that is not consulted at decision time. An administrator editing something labelled "policy" there changes nothing about authorisation.

11.5 Platform and operational limitations

11.5.1 One credential can degrade the rail

Partly closed. §10.4. Four synchronous workers are the entire clearance capacity; the database pool is not the binding constraint. Until 10 August 2026 there was no rate limit on the clearance endpoint and a single authenticated credential could occupy the rail. A per-credential rate and concurrency limit is now in force. **The underlying capacity — four concurrent requests on a two-core burstable instance — is unchanged**, and no fair-admission scheme distinguishes tenants.

11.5.2 There is no current capacity measurement

Deployment immaturity. §10.7. Earlier load figures predate the transaction and locking corrections and no longer describe the running system. No end-to-end latency, no sustainable clearance rate, no saturation point. For a rail sitting synchronously before every consequential action, this is an architectural property that cannot currently be stated.

11.5.3 Chain appends are globally serialised

Current architectural limitation. §10.7. Total ordering is correct for a hash chain, but a single serialised critical section is a scaling ceiling that adding workers does not raise. This will need addressing before any claim of national scale.

11.5.4 Evidence is stored in plaintext

Deployment immaturity. §10. There is no application-layer, column-level or host-level encryption of evidence payloads. Confidentiality at rest depends on whether the underlying cloud volume is encrypted — a control-plane setting outside the application, which protects against recovery of detached media and not against any party with database or operating-system access. Anyone with administrative access to the host, a database superuser role, or a copy of a backup reads the evidence in the clear. **The chain provides integrity, not confidentiality.**

11.5.4b A scan finding stored a fragment of the data it detected

Deployment immaturity — closed. §§5.4, 10.2. Until 19 August 2026 the output scanner wrote a six-character prefix of each matched value into its finding, labelled with the detected type and the PDPL article engaged, and stored it in `output_scan_log` — a table with no immutability trigger, on which the application role holds `UPDATE`, `DELETE` and `TRUNCATE`, and which is returned in full by an unscoped route. A live record held the leading digits of a Saudi National ID and of an IBAN. The field was removed from the writer and stripped from the three affected historical rows as a recorded operator action, with the before-state captured and hashed beforehand; the capture retains the removed values as an audit artefact.

No specimen entered the sealed evidence chain — verified against every payload, not assumed. The residual position is that the capture holds the only copies of the removed values, and that its replication off the host is manual rather than automated.

11.5.5 Secrets have no rotation procedure

Deployment immaturity. §10.3. No documented rotation exists for the session signing key, the token signing secret, the database password, the agent API keys, or the service credentials. Rotating the session or token secret today would invalidate every live session with no dual-key window. Automated rotation exists only for TLS certificates.

11.5.6 Historical backups contain plaintext secrets

Being closed. §10.6. From 10 August 2026 the local backup separates code from secrets, with the secrets artifact encrypted at creation to a key the host cannot read. **Twenty-three earlier bundles remain in plaintext**, each containing the session signing key, the token signing secret, the database password, fifteen TLS private keys, and **five live agent API bearer credentials** — for the legacy identities MASSAR-001, MAJLIS-002, MIZAN-003, MAARIFA-004 and MIDAD-005, each scoped for clearance, read and write.

The key type matters and is stated precisely. These are **shared secrets presented in a request header and matched against a stored hash. They are not signing keys**, and confer no ability to produce or forge a signed authorisation assertion. **No Ed25519 private key is present in the environment file, in any backup bundle, or anywhere on the rail's host**: MURAQIB stores only registered public keys, for six agents, and acts solely as a verifier. Exposure of these five credentials would permit impersonation of those agents at the authentication boundary — a credential incident requiring rotation — and **leaves the Rung 2 signature architecture intact**. Four of the five identities additionally hold no entry in the agent registry at all. They are retained only until the new arrangement is confirmed end to end, and the stronger remedy — rotating what they contain — is 11.5.5.

11.5.7 Credential and session weaknesses

Deployment immaturity. §10.3. API keys are stored as unsalted SHA-256 digests; the argument that this is adequate depends on generation entropy, which should be stated so the argument is testable. A query-parameter credential path existed until 9 August 2026. Cross-site protection on the clearance endpoint is the session cookie's same-site policy alone — no token, no origin check — which does not cover a same-site context.

11.5.7b Credential precedence discards the rejected credential

Deployment immaturity. §4.2. A request may present more than one credential, and resolution is by fixed order rather than conflict detection: a valid bearer token short-circuits the API-key path entirely, so the second credential is never read or validated, and an invalid bearer falls through to the API key with no record that it was presented. Token validation returns success or nothing — an expired and a forged token are indistinguishable to the caller and to the record. **A request can therefore authenticate as one**

principal having also offered a rejected credential for another, and the sealed record shows only the principal that succeeded. Nothing in the evidence establishes that a second identity was attempted.

11.5.7c There is no second authentication factor

Deployment immaturity. §10.3. Human authentication is a password alone, and this holds for every role including the most privileged: the account that creates users, issues and revokes agent credentials and reads across tenants is protected by a single reusable secret. No time-based code, hardware authenticator or out-of-band confirmation is implemented anywhere in the application, and no code path exists into which one could be dropped. Password storage is sound and login is rate-limited and failure-blocked, which raises the cost of guessing; neither touches a password that has been phished, reused or captured elsewhere. **The rail's administrative role can alter who is permitted to clear actions, and it is the weakest-protected privilege in the deployment.**

11.5.7d No impact assessment underpins the authentication design

Deployment immaturity. §10.3. The authentication factors in force were chosen without a recorded analysis of what an authentication failure or a bypass would cost. No impact assessment, threat model or risk rating for the authentication surface exists in the deployment. This is the input an assessor expects to find behind a single-factor decision — not the decision itself, but the reasoning that makes it defensible or shows that it is not. **Absent it, password-only access to the most privileged role is an assertion rather than a judgement**, and there is nothing on record to revisit when the exposure changes.

11.5.8 Three behavioural detectors have never run

Deployment immaturity. §10.6. A query comparing a text column to a timestamp without a cast fails, poisons the surrounding transaction, and prevents the two detectors after it from running, with every failure discarded silently. Rubber-stamp approval detection, financial-outlier detection and off-hours activity detection have therefore never produced a result, while the interface reported that no anomalies were found. **The interface wording was corrected on 9 August 2026; the detectors are not yet fixed, and a corrected off-hours detector would fire today on at least one agent.**

11.5.9 Sixty-two percent of the chain has no tenant attribution

Inherent boundary, given an immutable record. §10.5. Records predating attribution cannot be given it retroactively, because the records that would carry it are sealed. A tenant-scoped view excludes them and reports how many were excluded. **A tenant therefore cannot ask for the whole of its historical governed actions**; tenant-complete evidence begins at the attribution cutover.

11.5.9b Tenant-attribution failure permits evaluation to continue

Current architectural limitation. §6.1. When the tenant lookup fails, the rail records a sentinel and continues rather than refusing. This is defensible today because no control reads tenant identity and no binding is selected by it — tenant is a bookkeeping attribute, not an input to the verdict. **It ceases to be defensible the moment tenant identity participates in read isolation or authorisation**, at which point an unresolved tenant must fail closed or enter a non-clearable path. That transition is a required change, not an option.

11.5.10b Human password storage uses bcrypt over a pre-hash

Deployment immaturity. §10.3. Passwords are stored with bcrypt at cost 12 over a SHA-256 hexadecimal pre-hash, chosen so no password is truncated by bcrypt's input-length limit. The construction avoids the two known failure modes of pre-hashing — the digest is hexadecimal so contains no null byte, and it is used for no other purpose — but a memory-hard function would be the current recommendation for a new system.

11.5.10e MASSAR password storage: lazy migration to bcrypt, as defence in depth

Deployment immaturity, being reduced 15 September 2026. §10.3. MASSAR stored human passwords as a single pass of SHA-256 over a random 32-byte salt, with no work factor. All 34 accounts were stored this way, in both the live Postgres store and a frozen file copy, byte-identical between them.

This was hygiene, not a reachable exposure, and the change should not be read as closing one. Measured on the host: Postgres listens on loopback only, the users table grants to a single role, and the file store is mode 0400. The hashes were readable by `root` and by `masaar`, the account the application runs as — and either identity can already read the database those passwords protect, so no attacker reached a hash without first holding something better. What a stronger hash buys is resistance in the one case where the data outlives the boundary that protects it: an unencrypted nightly dump, a host snapshot, a disk handed on. Fourteen local `pg_dump` files containing the users table are retained on that host at any time. **The value of this change is defence in depth against a copy escaping, not the closure of an attack path that was open.**

The stored form now carries a scheme marker, `bcrypt$` or `sha256$`, and an unmarked value is still read as the original scheme so every existing account continues to work untouched. New hashes are bcrypt over a SHA-256 pre-hash, the same construction as 11.5.10b, **at cost 10 rather than that item's 12**. The lower factor is a deliberate consequence of the hardware and is stated so it is not read as an oversight: the host is a single vCPU running one gunicorn worker with one thread, so a password hash does not merely delay the person logging in — it serialises every other request for its full duration. Measured on that host, cost 12 takes 263 ms and cost 10 takes 66 ms. Cost 10 remains roughly four orders of magnitude more work than the single pass it replaces, and the factor should rise when the host gains cores rather than on principle.

Migration is lazy: a hash is upgraded in place on its owner's next successful login, which is the only moment at which a correct password and its stored hash are both in hand. **The upgrade writes both stores**, Postgres and the legacy file copy, in the same order and for the same reason as the credential tool that preceded it: a write path that touches one store is how a population silently diverges, which is precisely what happened to twelve MURAQIB credentials before 11.2.26. A mirror failure is logged explicitly and does not fail the login, so the worst case is a Postgres row ahead of the file copy with a record saying so, rather than a correct password refused. A forced reset was considered and rejected on the facts — 30 of the 34 accounts are deactivated and cannot reach the password check at all, and 9 have never logged in — so a reset would have been a reset nobody performed.

The accepted cost, stated rather than glossed: an account that never logs in again keeps its SHA-256 hash indefinitely. On current evidence that is most of the population. The four accounts that can authenticate hold 152 of the 214 recorded logins between them and would convert within a quarter at

their observed cadence. A second consequence is operational: once any row has been upgraded, reverting the application code would lock that user out, because the previous code cannot read either marker. The rollback is therefore not symmetric after the first successful login, and a revert must restore the stored hash alongside the code.

11.5.10c There is deliberately no break-glass path

Current architectural limitation, chosen. §3.6. No mechanism bypasses the clearance contract in an emergency. A break-glass path could be strongly authenticated, dual-controlled, time-limited and scoped — the objection is not that such a design is impossible, but that any bypass weakens the invariant that a governed action requires a rail verdict, and that invariant is what every claim in this document rests on. **The operational consequence is that availability failure cannot be administratively overridden**, and an institution requiring emergency continuity must address it through availability engineering rather than an exception.

11.5.10d The anchor watcher is scheduled twice

Deployment immaturity. §10.6. Two independent mechanisms schedule the staleness watcher, sending alerts to two different destinations. The staging step is idempotent so no harm results, but alerting is split and ownership is ambiguous.

11.5.10 The host is shared, and administrative access is broad

Deployment immaturity. §10.1. Six sibling applications and one application belonging to a separate system share the host, nine virtual host entries in all. SSH is internet-reachable, protected by key-only authentication and failure blocking, without an administrative bastion, source restriction or hardware-backed keys.

11.5.10f There is no privileged access management

Deployment immaturity. §10.1. Privilege is separated by role and by nothing else. No credential vault holds administrative secrets, no session recording captures what an administrator does, and no just-in-time mechanism grants elevation for a bounded period — administrative access is standing. Sudo is configured without session logging, and the administrative account holds unrestricted password-less escalation, which is conventional on a cloud instance and unrecorded when used. The seven application roles are a real separation of capability and are not a substitute for any of this: they govern who may act, and record nothing about what was done with the access once it was granted.

11.5.11 No idempotency

Deployment immaturity, and a prerequisite. §6.6. A retried request after a lost response is a new request, evaluated afresh and sealed as a second record. Nothing deduplicates concurrent identical requests. Unremarkable for a decision point; **materially serious under resource-side enforcement**, where two clearances for one intended action would authorise two executions.

11.5.12 Some failures return unstructured errors

Deployment immaturity. §§4.6, 6.3. An environment-resolution failure returns a different non-verdict token than the rest of the rail; an evidence-write failure returns a bare error carrying the raw exception text with no decision field; and the declared-summary scan has no error handling of its own, so an exception in it propagates out of the request and the caller receives an unstructured server error rather than the governed non-verdict. All three are fail-closed — the scan runs before anything is written, so a failure leaves no record at all — and all three give an operator less to work with. **The scan has no "could not classify" state**: content whose form it does not recognise returns clean, indistinguishable from a genuine clean result.

11.5.13 Off-box backup recoverability is unproven

Deployment immaturity, open. §10.6, A.2. The weekly restore drill exercises the locally retained plaintext artifacts and currently passes all seventeen checks. It does not exercise the encrypted off-box copies, and cannot: the originating host holds no private key and therefore cannot decrypt what it produces — the property that makes the off-box channel safe is also what leaves it untested. **End-to-end recovery from the off-box channel has never been demonstrated**, so a failure of the local artifacts would fall back to a path no rehearsal has covered. A decrypt-side rehearsal and custody documentation are scheduled.

11.5.13b Pin durability is inverted: the superseded pin is archived, the deployed one is not

Deployment immaturity. §§7.4, 10.4. `/home/ubuntu/oqiron-audit/` on the rail is covered by no automated backup or replication path. The nightly configuration bundle builds its file list from `/opt/muraqib` and named `/etc` paths, and the only `/home/ubuntu` content it captures is the SDK tree; the off-box push ships solely from `/opt/muraqib/backups` against four filename patterns, none of which this directory matches. Copies are taken by manual `scp` when an operator remembers; nothing schedules or verifies it. What the directory holds is a **superseded** policy-core pin with its custodian signature and its control-level manifest — a digest that is not deployed, was never sealed into the chain, and that no decision references — together with the pre-remediation capture recorded at 11.5.4b.

The exposure is the inversion, not the directory. The pin and custodian signature for the digest that is ACTUALLY DEPLOYED live only in `/tmp`, which does not survive a reboot, and **no control-level manifest for that digest exists in any location**. Two consequences follow and neither is hypothetical. A restart from cold would leave the deployed policy core with no retrievable signature, so `signed_status()` would report a correctly signed policy as unsigned — for reasons unrelated to any tampering, as 11.5.4-adjacent provenance items already note. And because the manifest is the input to the control-level diff, the next `POLICY_CHANGED` event will again carry *"control-level diff unavailable (no prior manifest archived)"*, exactly as the only sealed policy event does today. Archiving the superseded artefacts more durably would address neither.

11.5.14 There is no web application firewall

Deployment immaturity. §10.1. Requests reach the application through a reverse proxy that terminates TLS and applies rate limits, and that performs no request inspection: no module, no managed rule set, no signature matching, no anomaly scoring. The application's own input handling — parameterised queries, allow-listed payload construction, schema-bounded fields — is the entire defence. That defence is sound as far as it has been read, but it has not been tested adversarially, and there is no second layer that would catch what a reading missed.

11.5.15 Strict transport security is not asserted

Deployment immaturity. §10.1. Responses carry no `Strict-Transport-Security` header, so a browser that has once reached the service over TLS is never instructed to refuse plaintext on a later visit. The residual exposure is narrow — port 80 returns 404 rather than redirecting, so no plaintext session is ever served — but the header is precisely what defends the first request of a session, before any redirect can act. Frame options, content-type options and referrer policy are set; content security policy is not.

11.5.16 Logs are neither collected nor correlated, and the kernel is not audited

Deployment immaturity. §10.6. Application, proxy, database and system logs stay on the machine that produced them. Nothing forwards them and nothing aggregates them across the two hosts, so no correlation across sources is possible. Content matching exists but is narrow and self-referential: the five-minute watcher consumes the journal from a cursor and greps it for six named failure conditions the deployment emits about itself, and one nginx rate-limit pattern drives the login ban jail. **Nothing evaluates the security event log as a class** — there is no rule for an authentication anomaly, a privilege escalation, an unexpected process, or any condition not on that fixed list, and a marker the deployment does not emit is a marker nothing looks for. Kernel-level auditing is not installed, so privileged command execution, file access and configuration change leave no record independent of the operator performing them. **An administrator therefore holds the only copy of the evidence of what they did, on the machine they control.**

11.5.17 The journal retains approximately one hundred days

Deployment immaturity. §10.6. The system journal reaches back to 2 May 2026 — one hundred and one days at the time of writing — and is bounded by disk consumption rather than by a configured period; no retention time is set. This is the record carrying authentication attempts, privilege escalation and remote access, and it is the only such record. The window is not a decision anyone took; it is whatever 1.2 GB of journal happens to span, and it shortens as log volume grows.

11.5.18 Nothing detects malware or file tampering within the guest

Deployment immaturity. §10.1. No anti-malware scanner, host intrusion detection system or file integrity monitor is installed. Nothing would report a modified binary, an added service or a new file in the application tree. The evidence chain detects tampering with its own records and says nothing about the host that writes them.

11.5.18b No restriction on external storage media is configured

Deployment immaturity. §10.1. No kernel module for USB, FireWire or Thunderbolt storage is blacklisted or denied, no device-control software is installed, and no udev rule refuses a block device. Nothing would stop an attached storage device from mounting. The practical exposure is small — the host is a cloud instance carrying one NVMe volume with no removable device attached, and physical access to the machine is the provider's control rather than OQIRON's — but the control asks for a restriction and none is implemented. **The position rests on the environment rather than on anything configured, and it would not survive a move to hardware anyone can reach.**

11.5.19 There is no incident response plan

Deployment immaturity. §10.6. No documented procedure states who is notified when the rail is compromised, degraded, or found to have decided wrongly. There is no severity model, no escalation path, no stated response time, and no record type for an incident anywhere in the system. Operational alerting exists and works — a watcher on a five-minute timer sends mail on named conditions — but alerting is detection, and detection is not response. **Nothing has been rehearsed, so the first real incident would also be the first exercise of a process that does not yet exist.**

11.5.19b There is no notification path to a customer

Deployment immaturity. §10.6. Detection works and alerting works, and both terminate at an OQIRON mailbox. Nothing carries a detected condition to a customer: no customer-facing alert, no webhook, no status surface, and no contact address recorded against a tenant anywhere in the deployment. No notification timeline is defined either — nothing states how quickly a customer would be told, or by what route. **A customer learns of a detected condition by asking, or by verifying the chain itself.** Under a processing agreement obliging notification without undue delay, that commitment would rest on an operational undertaking with nothing built behind it.

11.5.20 There is no reporting path to the national authority

Deployment immaturity. No mechanism, integration or documented procedure exists for notifying a national cybersecurity authority of an incident, and nothing receives, evaluates or acts on threat intelligence. This is an obligation of the operating entity rather than a property of the architecture, and it is unmet. **The policy catalogue MURAQIB evaluates on behalf of its customers contains controls of exactly this kind.**

11.5.21 Conformance with the National Cryptographic Standards has not been assessed

Deployment immaturity. §§8, A.1. The algorithms in use are conventional and current: SHA-256 throughout the chain, Ed25519 for anchor signatures, bcrypt over a pre-hash for passwords, RSA-4096 with AES for backup encryption, and TLS 1.2 and 1.3 at the edge. Whether those selections, their key lengths and their handling satisfy the National Cryptographic Standards at the level this deployment would be assigned has never been checked against the standard's text, and no assessment record exists. **The statement here is neither conformance nor non-conformance — it is that the question has not been asked.**

11.5.22 There is no patch currency posture

Deployment immaturity. §10.1. Unattended upgrades are installed, enabled and running daily, so packages are applied without intervention. What is absent is everything around that: automatic reboot is not enabled, so upgrades requiring one accumulate until an operator acts; no maximum age is defined for a pending upgrade; no reboot window is scheduled; and no threshold exists at which a pending upgrade or an outstanding reboot becomes a finding. The monitoring watcher tests neither condition, so the state is visible only to someone who goes looking. **The count of pending upgrades on any given day is not the limitation — the absence of any defined point at which that count would matter is.**

11.5.23 A policy change sealed itself unsigned before the signature could be placed

Deployment immaturity, disclosed. §§7.4, 10.4. On 22 August 2026 the control catalogue was changed on disk at 23:34:58 while its custodian signature was still being produced offline. At 23:47:13 — **twelve minutes and fifteen seconds later** — a gunicorn worker recycled under `--max-requests`, re-imported `policy_core`, computed a digest that differed from the chain, and sealed `POLICY_CHANGED` at **seq 463 with `signed=false`**, logging `POLICY_CHANGE_UNSIGNED` at CRITICAL. No restart was issued and no operator action occurred; the recycle is routine and continuous. The signature was placed at 00:10:48, **twenty-three minutes after the record was already permanent**.

The trigger is a module import, not a restart, and the operating procedure said otherwise. The re-pin runbook in force at the time instructed the operator to place the pin before *restarting*. Measured on this deployment, a worker recycles roughly every sixteen minutes and gaps as short as one second have been observed, so the interval in which an unsigned policy edit can sit on disk unsealed is minutes, not a window the operator controls. The runbook has been corrected to require that the digest be computed from a copy, signed, and the pin placed **before the policy files are written**.

The record is permanent and is not to be repaired by reverting. `evidence_events` carries an append-only trigger; seq 463 cannot be amended or removed, and should not be. Reverting the catalogue would move the digest again and seal a second unexplained transition. The remedy applied was the only correct one: place the signature, confirm it covers the deployed digest, and disclose the gap. The deployed catalogue is signed going forward — `signed_status()` returns the pin — but **the chain records, permanently, that this change was made without one**. The strict setting described at §11.4 and D5 §4.4 would not have prevented it: the event is committed before the refusal is raised.

11.6 Institutional and scope limitations

11.6.1 OQIRON's own governance is not addressed here

Objective, and not yet published. §1.4. Ownership, neutrality commitments, the amendment process for the clearance standard, participant representation, independence of certification and resistance to capture are questions of institutional legitimacy. §3.4 establishes that they are also a security dependency: an operator holding the sole signing key is constrained by external retention, not by cryptography. A governance framework addressing this is planned and not written.

11.6.1b No control requirements are documented or approved

Deployment immaturity. §1.4. Across every control domain the deployment touches — identity and access, systems protection, data protection, cryptography, backup and recovery, logging, incident management, web application security and cloud — no document states the requirement, none has been approved, and there is nothing against which an implementation could be checked. What exists is the implementation and this document's description of it. The distinction is not clerical: an undocumented control that works cannot be shown to be the control that was intended, and a later change to it is a deviation from nothing.

11.6.1c Nothing is reviewed on a schedule

Deployment immaturity. §1.4. No periodic review of any kind is scheduled or recorded — not of access rights, not of agent credentials, not of policy content, not of configuration, not of the controls set out in this document. Several timers run unattended, covering backup, anchoring, staleness and the restore drill; none is a review, because each exercises a mechanism rather than reconsidering a decision. **Every entry in this register is therefore held at whatever state it was last left in, with no scheduled occasion on which it would be looked at again.**

11.6.1d No acceptable use policy is published

Deployment immaturity. §1.4. No acceptable use policy, terms of use, or usage policy is served to a user of the web interface; every route an assessor would check returns not-found. Users of a governance system are told in detail what is required of the actions it governs, and nothing about what is required of them in using it. This is a publication gap rather than a technical one, and it is the kind an assessor notices first, because it is visible without any access at all.

11.6.2 Data residency and sovereignty

Deployment immaturity. §10.1. Production runs on a commercial cloud instance in Frankfurt. The secondary machine — holding the encrypted off-box backups and running the independent watcher — belongs to a second provider, also in Frankfurt. Administrative access is exercised from outside the region. Nothing is deployed in the Kingdom, and the production machine additionally serves nine virtual hosts, among them an application belonging to a separate system. For the sovereign constituency named in §1.5 these facts determine whether the architecture is sovereign at all, and for a customer under a localisation requirement they are not a hardening gap that can be closed on the present footprint: they require a different deployment, and no in-Kingdom or on-premise deployment has ever been built or observed.

11.6.3 On-premise deployment does not exist

As built. §10.8. The claim that the clearance contract is identical on-premise is architectural, not observed. An on-premise deployment shifts key custody, ceremony discipline, and every operational control to the customer — and the single-custodian reasoning in §3.4 applies equally to the customer's custodian.

11.6.4 Evidence retention and personal data

Deployment immaturity. Sealed payloads contain human identities and business content, and the chain is immutable by design. No retention period, minimisation policy, legal-hold behaviour, or approach to erasure is defined. This is compatible with evidentiary integrity but requires design, and it is not designed.

11.6.5 Out of scope entirely

Inherent boundaries. §3.9. MURAQIB does not make an agent correct, does not establish that a declaration was true, does not secure the systems an agent acts upon, and does not address compromise of the host operating system, language runtime or dependencies. Four-eyes is a control against unilateral action, not against collusion.

11.6.6 Sub-processors are not disclosed

Deployment immaturity. Six third parties sit in the delivery path: two infrastructure providers hosting the production and secondary machines, a model provider, a transactional mail provider, a messaging provider and an outbound mail relay. No customer-facing list, processing agreement or privacy notice names any of them. One receives content rather than merely carrying it — the Arabic explanation path transmits declared action summary text to a model provider outside the Kingdom, and the ledger holds four hundred and thirty-nine entries carrying a non-zero token cost between 16 February and 8 August 2026. **This is a transfer that has occurred, not a capability that might be exercised.** That path is reachable only through the legacy clearance endpoint and not from the current one, which bounds its future use and not its history. MURAQIB's own policy catalogue contains controls requiring precisely this disclosure.

11.6.6b No third party has been risk-assessed

Deployment immaturity. No cybersecurity risk assessment of any third party in the delivery path is recorded anywhere in the deployment — no assessment document, no register, no table, no contractual artifact. The same six providers that go undisclosed have also never been evaluated, so neither OQIRON nor a customer can say what was weighed before each was relied upon, or on what basis a replacement would be judged. This is a procurement and governance activity rather than a technical control, which is why no amount of hardening closes it.

11.7 What has been corrected, and what remains open

Twelve defects were found during the preparation of this document. They are listed with their true status, because "found and closed" would be false for three of them, and by the manner of their finding, because that is part of what a reader should weigh.

Closed and proven by induced failure

A verdict could be returned without being sealed. The evidence write and the chain seal now commit as one transaction. §6.5. One production verdict was affected; disclosed in 11.2.8.

Fork protection had been silently disarmed. A connection-pool mode defect released the transaction-scoped lock immediately, so appends never serialised; the uniqueness constraint caught every collision. Repaired and measured serialising. §6.5.

A route returned an approval regardless of policy. Disclosed in full in 11.2.8.

An action class approved on unsigned four-eyes fields. Brought under the signature requirement 8 August 2026. Two historical approvals disclosed in 11.2.8.

An infrastructure failure was presented as a governance verdict. An evaluator failure now returns a non-verdict rather than an escalation, so an auditor can distinguish "policy determined a human must decide" from "policy never ran." No sealed record was affected; established by investigating and excluding the one record that matched the shape.

A risk-scoring signal was silently dropped when it could not be evaluated. Now labelled, with a conservative band and the signals named. §4.10.

A recorded regulatory flag contributed nothing to the score. Field-name mismatch corrected; eleven affected records disclosed in 11.2.8.

The published verifier reported a truncated record as intact. Corrected in version 1.1.0 and documented in ADVISORY-2026-001. §9.4.

A credential could be presented in a URL. Removed 9 August 2026; no occurrence found in the retained log window.

Mitigated, with residual exposure

The clearance endpoint formerly carried no rate limit. A per-credential rate and concurrency limit is now in force, so one credential can no longer occupy the rail. **The underlying capacity is unchanged** — four concurrent requests on a two-core burstable instance — and no fair-admission scheme distinguishes tenants. 11.5.1.

Plaintext secrets in the local backup. New backups separate code from secrets, with the secrets artifact encrypted at creation to a key the host cannot read. **Twenty-three earlier bundles remain in plaintext and the credentials they contain have not been rotated.** 11.5.6 and 11.1.6.

Found and still open

Three behavioural detectors have never executed. A missing cast fails one query, poisons the transaction, and prevents the two after it from running, with every failure discarded. The interface wording was corrected on 9 August 2026 so it no longer asserts that all agents are within normal parameters; **the detectors themselves are not fixed, and a corrected off-hours detector would fire today on at least one agent.** 11.5.8.

How they were found

Six by reading code against claims — the seal atomicity defect, the connection-pool defect, the always-approve route, the unsigned four-eyes path, the dead detectors, and the unapplied scoring signal.

Six by adversarial review of this document — the truncation defect in the published verifier, the absence of a per-decision catalogue digest, the infrastructure failure wearing a verdict, the absent rate limit, the plaintext backup secrets, and the credential in a URL.

Every closed fix was proven by inducing the failure, not by observing its absence. Where a guard could not be made to fail, it was not counted as working.

The second list is the reason this document exists in the form it does. **Six defects in a production governance rail were found by writing down precisely what it claimed and then checking each claim against the code.** A reader may reasonably conclude that a system which has been examined this way is better understood than one which has not — and equally reasonably, that a system needing this many corrections was not as well understood before.

Both conclusions are available, and we would rather a reader reach them from a complete register than from a partial one.

Appendix A — Key Fingerprints

This appendix lists the cryptographic key identities referenced in this document so that a reader may verify signed artifacts independently, without contacting OQIRON.

A.1 OQIRON Anchor Key

- **Purpose:** signs evidence-chain checkpoint anchors. This is the key against which the published verification tool checks anchor signatures.
- **Algorithm:** Ed25519 (minisign)
- **Key ID:** 5B3AB7A784F5A1A2
- **Public key:** `RWSiofWEp7c6W3rnHzNw6KdRPsp1GYIlXavSaRhTfaj7IDlXrnLflnE5`
- **Published at:** oqiron.ai/verify
- **Custody:** private key held offline by the custodian on controlled hardware and present on no OQIRON production host; signing is performed manually as part of the documented anchor ceremony.

Note: operator-independent verification requires a checkpoint retained externally by the verifier. Possession of this public key alone establishes that an anchor was signed by this key, not that any given checkpoint is the most recent one. See B.6(e) and ADVISORY-2026-001.

A.2 Backup Archive Encryption Key

- **Purpose:** encrypts nightly database, globals, configuration and secrets archives before off-box transfer. The secrets archive carries credential and TLS private-key material and is encrypted at creation.
- **Algorithm:** RSA 4096
- **Key ID:** 451D225458B8B5BD
- **Public key publication:** not currently published.

Custody: not formally documented; the private key's location is asserted in source comments but has not been independently established as of this document's date. A weekly automated restore drill runs against the locally retained plaintext artifacts and currently passes all 17 checks. That drill does not exercise the encrypted off-box copies: the originating host holds no private key and cannot decrypt them. End-to-end recoverability from the off-box channel therefore remains unproven, and a decrypt-side rehearsal together with custody documentation are scheduled for the next engineering phase.

A.3 Agent Signing Keys

Public keys for agents under Rung 2 signature enforcement are not published in this document. Signature enforcement currently covers six of eight production identities; the associated key material is subject to change as coverage completes. These keys will be published in the MURAQIB Integration Standard when Rung 2 coverage is complete.

Appendix B — Canonical Form Specification

This appendix specifies exactly how an evidence record is serialised into bytes prior to hashing, so that a third party may recompute content and seal hashes independently and confirm chain integrity without OQIRON's cooperation.

B.1 Scope

This specification describes the production writer for the `POST /api/v2/intercept` path, which is the sole producer of `EVIDENCE_CREATED` events. A legacy SQLite writer using a different algorithm and a different genesis seed exists in the codebase, reachable via the `POST /clear` endpoint. Its store holds 495 historical records, written between February and August 2026 and predominantly seeded test data; those records are not part of the chain specified here and cannot be verified by this specification. No request to `/clear` in the retained request logs reached that writer. Those logs begin 19 May 2026, whereas the writer's non-seeded records date from 16 May 2026, so they do not cover the period in which those records were created; the records themselves, not the request logs, are the authoritative account of that writer's output.

B.2 Hashed field set

Content hashing covers an allow-list of fields fixed at the time each record is written. Records written before a given schema epoch carry fewer fields: the allow-list widened from 25 to 28 to 31 to 32 fields as signature-provenance and assertion fields were added, and each record is sealed against the field set in force when it was written. Key count therefore varies across the chain and is not a validity criterion. A verifier recomputes from the stored payload as-is and must not assume any particular key count.

Two fields, `tenant_id` and `policy_digest`, are conditionally present: they are omitted entirely when absent from the source record, rather than serialised as null. All other allow-list fields are serialised as JSON null when absent, which is what makes the non-conditional field count invariant within an epoch.

Fields not on the allow-list are excluded from the hash. These include the explanations and the post-write-mutable fields `approved_by`, `approved_at`, `approval_note` and `pushed_to_majlis`. The table that holds them carries no append-only trigger and the application role holds `UPDATE` on it, so **those column values** can be altered in place without breaking a seal and without a superuser. A content hash attests to the decision as recorded at write time and says nothing about an approval or a downstream push applied afterwards. **This is a statement about the columns, not about the information.** A human resolution is sealed in its own later event (`DECISION_RESOLVED`), which is how an act that happens after the decision is recorded without rewriting it; `pushed_to_majlis` has no such event and is genuinely unsealed. 11.2.18 states the consequence.

B.3 Serialisation

Canonical JSON via the Python standard library, with keys sorted lexicographically, non-ASCII characters emitted literally, and no whitespace between tokens. The resulting string is encoded UTF-8. Arabic text is therefore hashed as UTF-8 bytes, not as escape sequences. Serialised key order is lexicographic and does not follow the allow-list's declaration order.

Equivalent to `json.dumps(obj, sort_keys=True, ensure_ascii=False, separators=(",", ":")).encode("utf-8")`. Note there is no `default=` handler: every value must already be a string, number, boolean, null, object or array before serialisation.

B.4 Scalar canonicalisation

All scalars are coerced to strings before serialisation. Numbers become exponent-free decimal strings with trailing zeros stripped and negative zero normalised to zero. Booleans become the strings `"true"` and `"false"`, not JSON booleans. Null is preserved as JSON null. Empty strings are preserved and not coerced. Coercion is applied recursively through nested objects and arrays. This prevents PostgreSQL's JSONB numeric normalisation from altering values on re-read.

B.5 Hashing and chain linkage

The content hash is SHA-256 over the canonical bytes, expressed as lowercase hexadecimal. The chain link is a second SHA-256 over the concatenation of the predecessor's seal hash and the current content hash, both as hexadecimal text, UTF-8 encoded — the hex is never decoded to raw bytes. The predecessor's hash is not a field in the payload and is not covered by the content hash.

For the first record in a chain there is no predecessor, and the genesis seed constant is substituted in its place: $\text{seal\_hash}_1 = \text{SHA-256}(\texttt{GENESIS\_SEED} \,\|\, \text{content\_hash}_1)$. The seed is treated as ordinary text under the same UTF-8 concatenation rule; it is not hashed beforehand and not decoded.

Events are iterated in ascending `seq` order. Each table is an independent chain with its own genesis seed: the production chain seeds from the constant `GENESIS-OQIRON-MURAQIB-V1` and the sandbox chain from `GENESIS-OQIRON-MURAQIB-SANDBOX-V1`. These are fixed public constants, deliberately not derived from any checkpoint digest. A verifier presented with rows in any other order will report a break.

On verification, the predecessor value advances on the recomputed seal rather than the stored one, so a record cannot validate itself.

B.6 Stated limitations

Five properties of this specification are disclosed because they bound what a hash match proves:

(a) Event type is not covered. The `event_type` value is stored alongside the record but is not part of the hashed payload. A content hash therefore attests to a record's contents, not to its classification.

(b) Timestamps are not normalised. The encoder applies no date handling; `created_at` is hashed as the string supplied by the caller. Two records denoting the same instant in different formats produce different hashes. Callers must pre-format timestamps; the specification does not enforce a format.

(c) Stored payload text differs from hashed bytes. The `payload_json` column is written with a non-canonical serialisation. A verifier must re-canonicalise from the parsed structure, not hash the stored text directly. Hashing the stored text will not reproduce the content hash.

(d) A valid hash does not imply a non-empty record. The writer asserts no minimum content; a record with an empty payload seals and verifies cleanly. Hash validity attests to integrity, not to substance.

(e) Internal consistency is not authenticity. Recomputing content and seal hashes proves only that the chain is self-consistent; a chain rewritten end to end and re-sealed verifies cleanly against itself. Authenticity requires a second step: compare the recomputed tip against the digest of the most recent signed `PERIODIC_TIP` checkpoint, and verify that checkpoint's signature under OQIRON's anchor public key. The corresponding private key is held offline by the custodian under documented ceremony and is not present on any OQIRON production host; of the checkpoint row, verification uses only the digest, the signature and a key identifier, and no column of that table holds key material.

The anchor public key is a public value, published at oqiron.ai/verify, which carries the key value, and embedded in the muraqib-verify package. The identifier below is the value stored in each checkpoint's `pubkey_id` column.

```
key id:     minisign-5B3AB7A784F5A1A2
public key: RWSiofWEp7c6W3rnHzNw6KdRPsp1GYIlXavSaRhTfaj7IDlXrnLflnE5
```

The signature is minisign's prehashed mode (algorithm tag `ED`): $\mathrm{Ed25519\_verify}(\mathrm{pubkey}_{32}, \mathrm{sig}_{64}, \mathrm{BLAKE2b\text{-}512}(\mathrm{signed\_bytes}))$, where signed_bytes is the 64-character lowercase hex digest encoded ASCII, with no trailing newline. Both the public-key and signature lines are base64 over a 2-byte algorithm tag, an 8-byte little-endian key id, and the key or signature body; the key id embedded in the signature must equal the key id in the public key. Plain Ed25519 over the digest, or a digest with a trailing newline, will not verify.

The cadence of signed tips bounds the window in which an undetected rewrite is possible. At the time of writing the chain carries 23 checkpoints — one `GENESIS_FREEZE` and 22 `PERIODIC_TIP`, the latest covering seq 456 — all signed under the key above, against a chain tip at seq 457. Events after the most recent checkpoint are expected and normal: they carry integrity, in that they verify against their predecessors, but not authenticity, until the next tip is signed. A nonzero unanchored tail is not evidence of tampering.

Appendix C — Glossary

This glossary defines the terms used in this document. Definitions are given in plain language for a general reader; where a term has a precise technical construction, that construction is specified in Appendix B rather than here.

Action. A single operation an AI agent proposes to perform that has an effect outside itself — sending a message, moving funds, publishing a document, calling an external service. MURAQIB governs actions, not the reasoning that produced them.

Agent. A software system that decides and acts with some autonomy on behalf of an organisation. In this document an agent is any client that submits actions for clearance.

Allow-list. A list that names what is permitted, so that anything not named is excluded by default. MURAQIB uses one to determine which fields of a record are covered by its integrity seal; a field the caller supplies but the allow-list does not name is silently dropped. See B.2.

Anchor. See Checkpoint.

Anchor key. The cryptographic key pair used to sign checkpoints. The private half is held offline by a custodian under documented ceremony and is not present on any OQIRON production host; the public half is published so that anyone may verify a signature without OQIRON's assistance. See A.1 and B.6(e).

`APPROVED`. A verdict permitting the action to proceed.

`BLOCKED`. A verdict refusing the action.

Canonical form. A single, unambiguous way of writing a record down as bytes, so that the same record always produces the same fingerprint regardless of who writes it or on what system. Without a canonical form, two parties hashing the same record would disagree. Specified in full in Appendix B.

Checkpoint. A periodically signed statement of the evidence chain's state at a given point — recording how far the chain extended and what its cumulative fingerprint was. Checkpoints are what convert an internally consistent chain into an authenticated one, because the signature is produced with a key held on no OQIRON production host. Also called an anchor.

Clearance. A decision, issued before an action is performed, on whether that action may proceed. A clearance is not advice: the decision is issued pre-execution and is sealed into the evidence chain whether or not it is honoured. MURAQIB does not itself execute or block the action — enforcement rests with the calling agent, and the supplied SDK fails closed, treating any non-approval, and any unavailable verdict, as "do not proceed". An agent that proceeds against a refusal leaves a sealed record of the refusal it received, outside its own control; the record does not by itself establish that the action was performed.

Content hash. The fingerprint of a single record's contents. See B.5.

`ESCALATED`. A verdict withholding permission pending human authorisation. The action does not proceed on an escalated verdict alone.

Evidence chain. An append-only sequence of records in which each entry incorporates the fingerprint of the one before it, so that altering any earlier record invalidates every record after it. This makes undetected partial editing infeasible. It does not by itself detect a wholesale re-sealing of the chain, which is what signed checkpoints exist to bound — see B.6(e).

Fail-closed. A design principle under which a system that cannot complete its safety function refuses the request rather than allowing it through unchecked. One documented exception is recorded at 11.5.9b. In MURAQIB, a decision that cannot be recorded and sealed is not issued: the request fails rather than proceeding unsealed.

Four-eyes. A control requiring that a second, distinct person authorise an action proposed by the first, so that no single individual can both initiate and approve a consequential act. MURAQIB enforces the distinctness test — an authorizer who also appears in the maker set, by either identity or email, fails it — over identities asserted by the calling system and bound by its signature. The signature establishes who made the assertion, not that the named individuals are accurate; MURAQIB has no independent identity source of its own.

Genesis seed. The fixed, public starting value used in place of a predecessor's fingerprint for the first record in a chain. Published in B.5.

Hash. A short fingerprint computed from data, such that any change to the data produces a different fingerprint and the original data cannot be recovered from it. For the evidence chain MURAQIB uses SHA-256; checkpoint signatures additionally use BLAKE2b-512 as a prehash (see B.6(e)).

Integrity. The property that a record has not been altered since it was written. Distinct from authenticity — see B.6(e).

Intercept. The point at which an agent submits a proposed action to MURAQIB and waits for a clearance decision before proceeding. Governance occurs here, before execution, rather than in review afterwards.

Maker-checker. See Four-eyes.

Policy. The set of rules against which a proposed action is evaluated. Each rule states a condition and the decision that follows when it is met.

Pre-execution governance. Evaluating an action before it happens rather than auditing it afterwards. The distinction is consequential: post-hoc audit records what went wrong, whereas pre-execution clearance creates the opportunity to refuse before the fact, and a record of the choice either way.

Provenance. The recorded origin of something — which policy version made a decision, which party asserted an authorisation, which key signed a checkpoint.

Sandbox. A separate environment for testing, maintained as an entirely independent chain with its own genesis seed. Sandbox records are never part of the production chain.

Seal. The act of computing a record's fingerprint and linking it to its predecessor, fixing it in the chain. See B.5.

Seal hash. The fingerprint that links a record to the one before it. See B.5.

Tenant. An organisation whose data and decisions are attributed to it distinctly within a shared deployment. Every governed decision since tenant attribution was enabled records the tenant it belongs to, and that attribution is sealed into the evidence chain; decisions sealed before that point (chain positions 1–272) carry no tenant attribution and cannot acquire it retroactively without breaking their seals. Read-scoping — restricting each tenant's queries to its own records — is not enforced: the machinery is built and proven and six routes returning payloads are wired, but twelve remain. Tenant separation is presently a property of the record, not of access control.

Tip. The most recent record in a chain.

`UNAVAILABLE`. Not a verdict. Returned when MURAQIB could not reach a decision — because it was unreachable, or because a decision could not be sealed. It must never be read as permission: under the fail-closed principle the calling agent treats `UNAVAILABLE` exactly as it treats a refusal. A decision that cannot be sealed is not a decision.

Unanchored tail. The records written since the most recent signed checkpoint. These have integrity but not yet authenticity. A non-empty tail is normal and is not evidence of tampering. See B.6(e).