MURAQIB Compliance Manual

D5 · Compliance manual · content of 2026-09-18 · source b49b1fe5

This manual is written for compliance, risk and audit readers. It describes what MURAQIB does, how a decision is reached, and what is recorded. Every claim in this document cites the source file and line that supports it. Where a capability is limited, the limitation appears beside the capability rather than in an appendix.

الملخص التنفيذي

ما هو MURAQIB

MURAQIB هو مسار مقاصة (clearance rail) لأفعال وكلاء الذكاء الاصطناعي. قبل أن ينفّذ الوكيل فعلاً ذا أثر، يُصرّح بهذا الفعل إلى MURAQIB ويتلقّى حكماً: موافَق عليه، أو محظور، أو مُصعَّد إلى قرار بشري. ثم يُختم الحكم والتصريح معاً في سلسلة أدلة (evidence chain) مترابطة تشفيرياً.

ما لا يفعله MURAQIB

لا توجد نقطة إنفاذ (enforcement point). لا يعترض MURAQIB الفعل مادياً ولا يمنعه. الوكيل يسأل، وMURAQIB يقرّر، والوكيل هو من يلتزم بالقرار. وكيل مُدمَج على نحو صحيح لا يمضي عند صدور حكم بالحظر لأنه مكتوب على ذلك؛ أمّا وكيل خبيث أو معيب الدمج فقد يتلقّى الحظر ويمضي رغمه.

كذلك يحكم MURAQIB على **التصريح لا على المحتوى.** لا يطّلع المسار على مضمون الفعل ولا يخزّنه. فالموافقة تعني أن الفعل كما صُرّح به كان مسموحاً، ولا تقول شيئاً عمّا أُرسل فعلاً بعد ذلك.

ما تثبته الأدلة

يُكتب كل حكم في سجل لا يقبل إلا الإضافة، يفرضه محرّك قاعدة البيانات نفسه: الحذف والتعديل والاقتطاع كلها مرفوضة. ولا يجوز لسجلّين أن يدّعيا السابق نفسه، فلا يمكن إحقام تاريخ بديل إلى جانب التاريخ الحقيقي. وتُختم السلسلة دورياً بتوقيع مُنتَج بمفتاح لا يوجد على الخادم.

والحكم والدليل يُثبتان في معاملة واحدة: إن أخفق الختم، تُلغى المعاملة ولا يصدر الحكم أصلاً. **فلا توجد حالة يوافق فيها MURAQIB على فعل ثم لا يسجّله.**

حدود ما نثبته

السلسلة **كاشفة للتلاعب لا مانعة له (tamper-evident, not tamper-proof).** لا يستطيع التطبيق تعديل سجل مختوم بأي حال، أمّا من يملك صلاحيات إدارة قاعدة البيانات فيقيّده الكشف لا المنع. وبين ختم وآخر، لا يُكشف الاستبدال الشامل إلا بالمقارنة مع آخر توقيع.

ولا تستطيع السلسلة أن تثبت أنها عُرضت كاملة. ولا توجد براهين اشتمال لكل مستأجر على حدة؛ فالسلسلة واحدة لكل بيئة.

يتضمّن هذا الدليل سجلاً موحّداً باثنين وثلاثين قيداً من القيود، مذكورة كل واحدة منها إلى جانب القدرة التي تحدّها، لا في ملحق منفصل.

نموذج النشر

MURAQIB منتج يُنشر داخل بيئة المؤسسة أو داخل مركز بيانات في المملكة، تحت سيطرة المؤسسة وولايتها القضائية.

الفحص المستقل

أداة التحقق منشورة برخصة مفتوحة، وتعمل دون اتصال بأي نظام تابع لـ OQIRON، وتُعيد تنفيذ قواعد التجزئة من المواصفة المكتوبة بدلاً من استيرادها — إذ إن أداة تحقّق تشارك النظامَ شيفرتَه لا تثبت شيئاً. غير أن **التأكّد مستقل، أمّا الحصول فليس كذلك**: حزمة الأدلة يُنتجها مشغّل، ولا يوجد مسار ذاتي للحصول عليها.

---

1. What MURAQIB is

1.1 The clearance model

An AI agent that can take consequential action — send correspondence, move money, grant access, publish a statement — poses a question no logging system answers: **by what authority did it act, and who can prove that afterwards?**

MURAQIB answers it with a clearance model borrowed from financial market infrastructure. Before an agent performs a consequential action, it declares that action to MURAQIB. MURAQIB evaluates the declaration against a fixed catalogue of controls and returns one of three verdicts: the action is approved, it is blocked, or it is escalated to a human. The verdict and the declaration are then sealed into a hash-chained evidence record.

Three properties follow, and they are the substance of the product:

The decision is made before the action, not discovered after it. An institution reviewing an agent's behaviour is not reconstructing intent from logs; it is reading a decision that was made at the moment of the action, by a system whose rules were fixed in advance.

The sealed decision cannot be quietly revised. Records in the evidence chain are append-only at the database level. A sealed record cannot be edited or deleted by the application under any circumstance, and the chain's structure makes an alternative history impossible to graft alongside the real one. The same decision is also written to a second, unsealed operational store, which carries no such trigger and which the application may update in place; that store, not the chain, is what the dashboard and the spreadsheet exports read (§5.0). Section 5 explains precisely how, and Section 6 explains precisely what this does not cover.

The decision is verifiable by someone who does not trust OQIRON. A public, independently written verification tool checks an evidence bundle offline, without contacting any OQIRON system. Section 10 covers this, including the part of it that is not yet independent.

1.2 What MURAQIB is not

There is no enforcement point. This is the most important sentence in the manual and it is placed first deliberately.

MURAQIB does not sit in the network path of an agent's action and physically prevent it. The agent asks; MURAQIB rules; **the agent honours the ruling.** A correctly integrated agent does not proceed on a BLOCKED verdict because it is written not to. A deliberately malicious agent, or one whose integration is defective, could receive a BLOCKED verdict and act anyway.

What MURAQIB guarantees is that the verdict was rendered by a fixed policy, that it was rendered before the action, and that the record of it exists and cannot be altered. What it does not guarantee is that the calling system obeyed.

This places MURAQIB in the same category as pre-trade compliance checks, sanctions screening services and credit bureaux: authoritative decision services whose output an executing system is obliged to respect, and whose value lies in the authority and the record rather than in physical interception.

MURAQIB governs the declaration, not the payload. The rail never sees, parses or stores the content of an action. An approval states that the *declared* action was permitted. It says nothing about the bytes the agent subsequently sent. An agent that declares an action accurately and then does something different has produced an accurate record of a declaration and an inaccurate record of reality — and no clearance system that does not inspect content can detect that.

This is a deliberate design choice with a direct privacy consequence: MURAQIB can govern correspondence without reading correspondence, and can govern a transfer without holding account details.

MURAQIB is not an accredited certifier. Evidence is sealed and offline-anchored. That is a cryptographic property. It is not a regulatory attestation, and no regulator has accredited it.

1.3 Where it sits in a control environment

MURAQIB is a control that operates on agent actions. It is not a substitute for the controls an institution already runs on its systems, its data, or its people. It answers one question — *was this agent action permitted, and can that be proven* — and it answers nothing else.

Institutions that already run pre-execution controls on human-originated activity will recognise the shape. MURAQIB extends the same discipline to actions originated by software that reasons.

1.4 Deployment model

MURAQIB is an on-premise and in-Kingdom product. The clearance rail, the policy core and the evidence chain are designed to run inside the institution's own environment or inside a Saudi data centre, under the institution's control and jurisdiction.

[Section 9 states what has and has not been exercised operationally, including the fact that a second production instance has never been built.]

---

2. How a decision is made

This section is the core of the manual. A compliance reader who understands only this section understands the product.

2.1 The path of a single clearance call

An agent's declaration passes through eleven stages before a verdict is returned. They run in a fixed order in a single request handler (`app.py:1715–2447`).

	Stage	What happens
1	**Authenticate**	The caller proves identity by token, API key or session. An unauthenticated request receives no verdict.
2	**Parse**	The declaration is read. An oversized summary is **refused, never silently truncated** — a truncated declaration would produce an accurate record of an inaccurate statement.
3	**Classify sensitivity**	The declaration is assessed for data-sensitivity characteristics.
4	**Resolve four-eyes fields**	Where the action carries an authorisation assertion, the asserted identities are resolved and any signature is verified.
5	**Derive server-side facts**	The agent's registration, scope, freeze state, tenant and **environment** are read from the server's own records — never from the caller's claims.
6	**Route by environment**	Production and sandbox declarations are routed to structurally separate evidence chains with different genesis lineage.
7	**Legacy aggregation**	A historical evaluation runs. **It is not authoritative** and is retained only for comparison.
8	**Clean core evaluates**	The authoritative policy evaluation. It **overrides** the preceding stage in all cases (`app.py:2028`).
9	**Output scan**	A safety overlay examines the response.
10	**Principal check**	A second overlay handles unregistered principals.
11	**Seal**	The decision and its declaration are written into the evidence chain (`app.py:2280`).

Stages 9 and 10 run **after** the policy verdict and can only make the outcome stricter, never more permissive (`app.py:2035–2037`). A safety overlay cannot turn a BLOCK into an APPROVE.

2.2 What determines the verdict

The declared action type determines which controls apply. MURAQIB recognises 103 action types. Each maps to a canonical class, and the class selects which of the 58 controls are relevant to the declaration (`policy_core.py:479`).

Matching is exact. There is no fuzzy matching and no nearest-neighbour suggestion. An unrecognised action type does not fall through to permission — it escalates (Section 3.1).

Every relevant control votes, and the strictest vote wins. The aggregation is ranked: BLOCK outranks ESCALATE, which outranks APPROVE (`policy_core.py:488–496`). A single BLOCK from a single control blocks the action regardless of how many other controls approved it. There is no scoring, no threshold and no majority.

2.3 What does not determine the verdict

This is the property most often misunderstood, and it is the one a compliance reader should test hardest.

An agent's declaration includes fifteen attribute fields — whether the action is external, whether it contains personal data, a financial amount, a priority, a regulatory-risk marker, and others. **None of them affects the verdict.** They are sealed into the evidence record and form part of the audit trail. They do not change the decision.

This is not a convention that a future change might quietly break. The field carrying the caller's claims is annotated in the source itself:

> `client_claims: dict` — **`# NEVER read for enforcement`** (`policy_core.py:191`)

and the evaluator's own description states its input boundary explicitly:

> *"Reads ONLY action_type/server/config/feeds.* `req.client_claims` *is never accessed here."* (`policy_core.py:475–476`)

The consequence matters for an institution assessing manipulation risk: **an agent cannot argue its way to an approval.** Declaring an action low-risk does not make it approved. Omitting a financial amount does not avoid a block. What the agent says about its action is recorded as evidence and is not believed as input.

What does decide: the declared action type; server-derived facts about the agent — its registration, scope, freeze state, environment, action velocity, and the server's own clock; and, where applicable, four-eyes attestations.

2.4 The five possible responses

A caller receives one of five responses. **Only three are verdicts.**

Response	HTTP	Sealed into the chain	Meaning
`APPROVED`	200	**Yes**	A governance decision. The declared action is permitted.
`BLOCKED`	200	**Yes**	A governance decision. Refused, and terminal.
`ESCALATED`	200	**Yes**	A governance decision. A human must decide.
`REFUSED`	503	No	Preconditions were not satisfiable. Policy did not run.
`UNAVAILABLE`	503	No	Infrastructure failed. Policy did not run.

The distinction between the first three and the last two is the distinction between *a decision was made* and *no decision was made*. Both prevent the agent from proceeding. Only one of them is evidence.

`ESCALATED` **is not "pending."** There is no endpoint that resumes an escalated action, no callback and no polling mechanism that later flips it to approved. The action stops. A human reviews the sealed record out of band. If the work proceeds, it proceeds as a **new** clearance call — a fresh declaration, not a continuation.

2.5 Why a non-decision is not sealed

An earlier version of MURAQIB returned `ESCALATED` when the infrastructure failed. The caller could not proceed, so the behaviour was safe. It was nonetheless changed, and the reason is recorded in the source:

> *"That was safe — the caller still could not proceed — but it was DISHONEST… An auditor reading a sealed ESCALATED record could not tell the two apart."* (`app.py:2057–2062`)

> *"A non-verdict is not a decision and must not enter the decision chain, so NOTHING IS SEALED here."* (`app.py:2064–2065`)

An institution's auditor can therefore rely on a property that is easy to state and was not easy to build: **every record in the evidence chain is a decision that a policy actually rendered.** Infrastructure failures are logged and alerted. They do not become evidence of governance that did not occur.

---

3. Fail-closed behaviour

3.1 The five failure modes

MURAQIB fails closed in every case. A caller that cannot obtain a verdict is not permitted to proceed.

Condition	Response	Reference
The declared action type is not recognised	**ESCALATE** under control `TAXO-00`	`policy_core.py:484–486`
The agent's environment cannot be derived server-side	`REFUSED` 503, before any write occurs	`app.py:1937–1941`
The database connection pool is exhausted	`UNAVAILABLE` 503 with `Retry-After: 2`	`app.py:123–138`
The policy engine raises an error	Transaction rolled back, logged at CRITICAL, `UNAVAILABLE` 503	`app.py:2054–2092`
The evidence seal fails	**Transaction rolled back — including the decision — and `UNAVAILABLE` 503**	`app.py:2284–2301`

The first row deserves emphasis for an institution assessing coverage. An action type MURAQIB does not recognise is **escalated to a human**, not approved. The system refuses to guess what an action it does not understand might be permitted to do. This means an incomplete taxonomy produces friction rather than silent permission.

Behaviour under sustained load, measured 5 September 2026. The clearance endpoint was driven from a single external client at one to sixteen concurrent connections, ten seconds per level, against the sandbox environment. 109 requests were served and **approved** at one connection. At two connections a further 11 were approved and the next 64 escalated. Every request served at four, eight and sixteen connections escalated — 225 escalations in total, under control `AGT-04` (*Velocity limit*, agent-control, tier A).

`AGT-04` is cumulative, not rate-based. Its threshold is a count of an agent's actions in a rolling hour — `action_count_hour > 120`, counted server-side — not a speed. It fires on the 121st action of an hour whether those arrive in ten seconds or fifty minutes, and it continues to fire until the hour's count falls below the threshold. The transition observed is exactly this: 109 approvals plus 11 more reached 120, and escalation began at the 121st request. **Concurrency did not cause the escalations; the accumulated count did**, and the same escalation would occur at any rate once the hour's total is passed.

The system became **more** restrictive as load accumulated, not less. No request was refused by the application: no `503`, no timeout, and the connection pool was never exhausted. Every rejection was a `429` from the reverse proxy, which is a distinct response from the application's `UNAVAILABLE` (§3.1 table). 345 clearance records were sealed during the measurement and the chain verified intact afterwards, every record reading `ok`.

3.2 No approved-but-unrecorded state

The verdict and the evidence record commit in a single transaction.

If the chain append fails for any reason, the transaction rolls back — and the rollback discards the decision along with the seal. The caller receives `UNAVAILABLE`. No verdict is issued.

The consequence is worth stating precisely, because it is the strongest property in the system:

> **A verdict a caller received is a verdict that was sealed.** There is no state in which MURAQIB approved an action and failed to record it.

This closes the gap that makes most audit logging weak. A system that decides first and logs afterwards can approve an action and lose the record — through a crash, a full disk, or a failed write — and no subsequent audit can distinguish that from an action that was never declared. MURAQIB cannot enter that state.

3.3 A caller cannot read absence as permission

Every failure response sets all three verdict flags to false (`app.py:129–131`):

```
approved: false
blocked: false
escalated: false
```

A client that checks only "was it blocked?" will not mistake an infrastructure failure for an approval. The published client library enforces the same discipline from the other side: every client-side failure resolves to `UNAVAILABLE`, and the library provides no fail-open mode.

3.4 What an institution experiences during an outage

If MURAQIB is unavailable, **every governed agent action is blocked.** No action receives permission.

This is the correct behaviour for a governance control and it is the behaviour an institution should expect. It is also an operational consequence that must be planned for: an agent estate that depends on MURAQIB for clearance stops taking consequential action when MURAQIB stops answering.

The reference deployment runs on a single host with no failover. An outage of that host is a total outage of the rail. Section 9 describes the operational controls that exist — monitoring, daily backup, off-box replication and a weekly restore drill — and the ones that do not. An institution deploying MURAQIB should treat availability architecture as part of its own deployment design.

---

4. The policy core and its integrity

4.1 What a control is

MURAQIB evaluates declarations against a catalogue of **58 controls**. A control is a rule that examines a declared action type together with server-derived facts and returns one of three votes: approve, escalate, or block. It has an identifier, a name in English and Arabic, and a stated reason that is sealed with the decision.

Controls do not consult the caller's claims (Section 2.3). They consult the action type, the agent's server-side record, configuration, and — where relevant — the four-eyes attestation.

A reader assessing coverage should note what this design trades away. A catalogue of 58 controls over 103 recognised action types is deliberate and finite. It does not attempt to reason about novel situations. An action type outside the catalogue escalates to a human rather than being assessed by inference (Section 3.1). **MURAQIB is a rule engine with a fixed catalogue, not a model that judges.**

4.2 The system refuses to start in a degraded state

Most systems degrade when a dependency is missing. MURAQIB refuses to run.

If the policy set is anything other than the authoritative clean core, the application raises at import and does not start (`app.py:59–63`). The source states why the older engine is not treated as a fallback: it contained 212 controls that read fields which were never populated — controls that appeared to be

governing and were not.

The same behaviour applies to three further dependencies. The agent registry (`app.py:67`), the credential store (`app.py:72`) and the user store (`app.py:78`) each cause a startup refusal if unreachable **or empty**. There is no in-code fallback to a default, an empty set, or a permissive mode.

The operational effect is that a failed start never becomes a running system with silently absent governance. The health endpoint does not return success, and the deployment automatically rolls back to the previous version.

For an institution's change-management assessment this is the relevant property: a misconfigured MURAQIB does not run permissively. It does not run.

4.3 How policy change is detected

The policy core is protected by a digest. Two files — the evaluator and its adapter — are canonicalised and hashed with SHA-256 into a single value (`pin_core.py:86–94`). That digest is sealed into **every decision record** as `policy_digest` (`app.py:2262`).

At startup, MURAQIB compares the current digest against the last one recorded. If they differ, a `POLICY_CHANGED` event is sealed into the evidence chain. If that seal cannot be written, the application does not serve (`app.py:105–109`).

A change to the rules therefore cannot take effect silently. Either the change is recorded in the tamper-evident chain, or the system does not run.

The mechanism distinguishes two sources of a recorded change (`policy_provenance.py:11–19`). A **ceremony** event is created deliberately by an operator and carries the name of the person who authorised it. A **startup** event is created by the system noticing a difference and can never carry an authoriser. The distinction gives an auditor a specific thing to look for: *a startup event with no preceding ceremony event for the same digest is an undeclared change.*

4.4 What this mechanism does not do

The module states its own boundary, and the manual states it in the same words:

> *"It records. It does not authorise, does not add approval or role separation, and does not stop anyone with write access to /opt/muraqib from changing policy — including by editing this file. It is detection, not control."* (`policy_provenance.py:21–24`)

An unsigned policy change is permitted to serve. A change to the control catalogue that carries no custodian signature is detected, sealed into the evidence chain as a `POLICY_CHANGED` event marked unsigned, and logged at CRITICAL. The reference deployment permits it to serve.

An opt-in strict setting exists and is **not yet effective as a preventive control.** It refuses a single start attempt and does not survive the automatic service restart that follows, because the change is committed to the chain before the refusal is raised — so the next start observes no change, does not reach the

signature check, and serves. **It must not be relied upon to prevent an unsigned policy change from taking effect.**

A related limit applies to the underlying check. The signature test is reached only on a start where the digest has changed. On an ordinary start it is unreachable, and the system therefore holds **no steady-state assertion that the policy currently deployed is signed**. Converting this from a change alarm into a state invariant is scoped work and is not complete.

This is not hypothetical. On 22 August 2026 a catalogue change was sealed at `POLICY_CHANGED` seq 463 with `signed=false`, twelve minutes after the files were written and twenty-three minutes before the custodian signature was placed. The seal was triggered by a routine worker recycle, not by a restart, and no operator action intervened. The change served from that moment. The signature now covers the deployed digest, and the unsigned record remains in the chain permanently. Whitepaper §11.5.23 records it in full. It is a direct illustration of the sentence above: **the strict setting would not have prevented it, because the event is committed to the chain before the refusal is raised.**

The module's own summary applies to strict mode as much as to the rest: *"It records. It does not authorise… It is detection, not control."* (`policy_provenance.py:21–24`)

4.5 The current state of policy provenance, stated plainly

Four limits apply to the reference deployment and would apply to any deployment for the period before the mechanism was installed.

Policy provenance begins on 8 August 2026. The first of the two recorded `POLICY_CHANGED` events carries `prev_digest = null`, because there was no prior recorded digest to compare against. **No policy change before that date is evidenced.** The manual does not claim continuous provenance and an auditor should not infer it.

That first event is marked retroactive. The mechanism was installed after the change it records. It is well-formed — it carries an authoriser, it is signed, and it states its reason — but it documents a change that had already occurred rather than one it observed.

`policy_digest` appears on 20 of 453 decision records. The field was added at the 8 August re-pin. Decisions before that date carry 33 fields rather than 34 and **cannot be tied to a specific version of the policy**. The accurate claim is that every decision records the policy in force *from 8 August 2026 forward.*

The record that a policy change was signed is only partly durable, and the control-level record is absent. Signature pins are written to a temporary location that does not survive a reboot; the pin and custodian signature for the currently deployed digest have been copied to durable storage as a recorded operator action, but nothing schedules that copy and no automated path replicates either location off the host. A deployment restarted from cold before such a copy is taken reports a correctly signed policy as unsigned, for reasons unrelated to any tampering. A control-level manifest has since been archived for both recorded digests, and the second sealed policy event (seq 463, 22 August 2026)

consequently names the control that changed — `added: ["AGT-08"]` — where the first could only state that a diff was unavailable. Whitepaper §11.5.13b records the durability inversion; §11.5.23 records the unsigned seal at seq 463.

---

5. Evidence: what is recorded

5.0 Two stores, one record

A governed decision is written to two tables in a single transaction. The **evidence chain** is the sealed, hash-linked, append-only store — the store the verification tool reads, the anchoring ceremony covers, and this section's integrity claims describe. The **operational store** holds the decision's content: risk score, rule fired, explanations, SAAC code, tenant identifier. **From 16 September 2026 the reasoning behind a decision is also sealed**, as a machine-readable trace of the signals examined and the rules that fired, carried in the chain record itself (§5.1). Before that date the chain proved *what* was decided and the operational store alone held *why* — and the operational store is mutable. Decisions taken before 16 September 2026 cannot acquire a sealed explanation, for the same reason no other field can be backfilled. Both rows carry the same evidence identifier, and a failure to seal rolls back the operational write, so no governed decision taken since 12 June 2026 exists in one store without the other.

Two consequences an institution should understand. First, **the dashboard and the spreadsheet and PDF export routes read the unsealed operational store**, on which the application holds update permission and which carries no immutability trigger. The offline evidence bundle reads the sealed chain and refuses to emit a file whose seals it cannot re-verify. Second, **the operational store predates the chain.** It holds 1,073 rows against the chain's 453 sealed decisions; **620 rows correspond to no sealed decision** — 563 from before sealing began on 12 June 2026, 56 from a superseded identifier scheme, and one from a load test in which a seal was lost to a known concurrency defect. The verifier classifies these as `legacy_unverifiable` and reports their count rather than silently including them. They are covered by the signed genesis checkpoint fingerprint, not per-row.

The twinning applies to decisions, and to nothing else. A reader who takes "two stores, one record" as a general property of the chain will draw the wrong conclusion about everything else in it. The operational store is shaped for decisions — its columns are agent, action type, risk score, risk level, decision, SAAC code and tenant — and it holds **no row of any kind** for a governance act. Of the eleven event types in the chain, **nine 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, and `DECISION_RESOLVED` attaches to a decision row that already exists.

Two consequences follow for an institution. An export from the operational store is a complete record of decisions and **contains no governance act at all**, so it cannot be read as an administrative audit trail. And the 620 unsealed legacy rows discussed above are unsealed *decisions*; there is no corresponding population of unsealed governance acts, because governance acts have only ever existed in the sealed chain or nowhere.

The chain's properties are unaffected by any of this. What this manual previously lacked was not integrity but a statement distinguishing the two stores.

5.1 The fields

Decision records seal at 38 fields from 16 September 2026. No record of that shape exists yet at the reference deployment: the most recent decision carries 34, and the first 38-field record will be created by the next governed decision. Older records carry fewer — 25, 28, 31, 32, 33 and 34 fields respectively — because fields were added as the system developed and a sealed record cannot be amended to add them. The immutability that protects the chain also prevents backfilling. An institution examining historical records should expect a narrower field set the further back it looks, and should read the absence of a field as *not captured at the time*, rather than as a null value.

The distribution across the 453 decision records, as of 16 September 2026:

Fields	Records
25	198
28	5
31	67
32	160
33	3
34	20
38	0

The 38-field row stands at zero deliberately. The four fields that define it were added on 16 September 2026 and seal from the next governed decision onward; the row is shown empty rather than omitted so that the first record to appear in it is a change an auditor was told to expect. The same table one week hence is the check.

These populations move in only one direction. A sealed record cannot be amended, so no record ever changes row; ordinary operation adds records at the newest shape and leaves every other row fixed. **A change in any row but the last would mean sealed records had been altered** — which makes this table a check an auditor can apply directly, rather than a statistic.

The fields of a current record, grouped by purpose:

Identity of the record — `evidence_id`, `request_ref`, `created_at`, `tenant_id`

Who acted — `agent_id`, `agent_name`, `actor_id`, `actor_name`

What was declared — `action_type`, `action_summary`, `priority`

What was decided — `decision`, `rule_triggered`, `rule_name_en`, `rule_name_ar`, `requires_approval`, `safe_action`, `risk_level`, `risk_score`, `regulatory_risk`

Declared attributes, recorded as evidence and not used for enforcement — `data_sensitivity`, `contains_pii`, `is_external`, `has_attachment`, `financial_amount`, `token_cost`

Which policy decided — `policy_digest`

Four-eyes material, where applicable — `assertion_body`, `assertion_sig`, `assertion_key_id`, `assertion_ts`, `assertion_bind_version`, `signature_verified`, `signature_reject_reason`

5.2 What is not recorded

Content is never captured on the clearance path. When an agent declares an action for clearance, the rail does not receive, parse, store or transmit the substance of that action. A record of a governed email contains the fact that an email of a declared type was proposed, to a declared category of recipient, and what was decided. It does not contain the email.

One path is different, by design. The declared-summary scanner receives the text of a proposed output and parses it, in order to detect personal data before that output leaves. It stores no content: the scan log retains a finding count, a risk level, a redaction flag, and the lengths of the original and redacted text. Until 19 August 2026 it additionally stored a six-character prefix of each matched value; that column has been removed and no specimen remains. The scan log is not part of the evidence chain, carries no immutability trigger, and the application holds full privileges on it including delete.

This has a direct consequence for a data-protection assessment: MURAQIB can govern correspondence without processing correspondence. It also has a consequence an auditor must understand — an evidence record proves what was *declared and decided*, not what was *done*.

5.3 How the chain resists alteration

Four mechanisms operate together. The second is the one worth reading twice.

Append-only at the database level. A trigger on the three chain tables — the production chain, the sandbox chain and the checkpoint table — raises unconditionally on DELETE, UPDATE and TRUNCATE. It does not extend to the unsealed operational store described at §5.0. INSERT is the only permitted operation. This is enforced by the database engine, not by application logic, so it binds any code path — including a defective or malicious one.

No two events may claim the same predecessor. Each record stores the seal of the record before it, and that column carries a uniqueness constraint. **This is what makes the chain fork-proof.** An attacker who wanted to present an alternative history could not simply write a competing branch alongside the real one: the second record claiming a given predecessor is rejected by the database. There is one lineage, and there can only ever be one.

Every seal is unique, and every record is complete. Uniqueness constraints on the seal and on the event identifier, and NOT NULL on all nine columns, remove the classes of ambiguity an auditor would otherwise have to rule out by inspection.

Offline anchoring. Periodically, the chain's current fingerprint is signed with a key that is held on a machine separate from the rail and never present on the server. Section 5.5 describes the ceremony.

5.4 A gap in numbering is not a gap in the chain

The sequence numbers on evidence records are not contiguous. The reference deployment's chain holds 495 records numbered up to 534, with 39 values absent.

This is expected and does not indicate loss. Sequence values are allocated by the database and are consumed when a transaction draws one and then rolls back; a rolled-back transaction burns a number and writes nothing. The reference deployment's 38 absent values have two distinct causes, stated separately because they are not the same event.

Twenty-one, all within a two-second window on 4 August 2026, were consumed during a load measurement in which a connection-pool defect allowed autocommit to leak across pooled connections, disarming the transaction-scoped advisory lock and losing appends. Those numbers were not burned by ordinary fail-closed refusals; they were burned by a defect, which has since been corrected and is recorded in the whitepaper's register.

Seventeen, spanning 29–30 August 2026, were consumed by engineering drills and by proof runs executed against live files while a new event type was being introduced. They record no governed decision of any kind. They were deliberately not reset: a sequence reset would leave no trace, and the record would then be tidier than the truth.

Records are linked by the **stored seal of the preceding record**, never by arithmetic on the sequence number. Verification tests each record against the record immediately before it in order, not against the record numbered one less. On the reference deployment: **443 records, 442 links, zero broken.**

All counts in this manual are stated as of 19 August 2026. **They will diverge from the live system, and that divergence is the expected consequence of ordinary operation** — the chain grows whenever the rail does its job, so every cleared action moves these figures by one. A consistency checker published with this manual verifies each figure against the live system and fails when the two no longer agree. **A failing checker therefore reports that the manual is due for reissue. It does not report that anything is wrong with the deployment**, and the correct response is to reissue the manual — never to amend the checker to match.

An auditor examining the numbering should test the links rather than the numbers, and the published verification tool does exactly that.

5.5 Anchoring

An anchor is a signature over the chain's state at a point in time, made with a key that does not exist on the rail.

The ceremony has three stages and a genuine separation of duties (`ANCHOR_RUNBOOK.md`). The system stages a read-only digest and **never signs**. The operator signs that digest offline on a separate machine; the key and its passphrase are theirs alone. The signature is then inserted by a guarded script that verifies the signature before opening a database connection, re-derives the chain state to confirm it still matches what was signed, and refuses a duplicate.

The purpose is to place a limit on what someone with full control of the server could do. Such a person could rebuild the entire chain so that it was internally consistent. They could not produce a valid signature over it, because the key is not there.

Current state of the reference deployment: checkpoint 24 is the most recent signed checkpoint. Records sealed after it are chain-linked but not yet under an offline signature; §5.6 states what that means. The gap between the chain tip and the latest checkpoint varies with activity and closes at each ceremony.

5.6 Anchoring is periodic, not immediate

A record is sealed and chain-linked **at the moment of the decision**. It comes under an offline signature **at the next ceremony**.

Between ceremonies, the most recent records are chain-linked but not yet anchored. An institution should read this precisely: those records are protected by the append-only trigger and the fork-proofing constraint, and they are not yet protected against wholesale substitution by someone able to rebuild the chain.

The ceremony is operator-driven. The reference deployment monitors staleness daily and stages automatically, but a human must sign.

---

6. The limits of the evidence

These limits appear here together because they concern the same subject. Each is also stated beside the capability it qualifies.

6.1 Tamper-evident, not tamper-proof

The application connects to the database as a role that is neither the table owner nor a superuser. **It cannot disable the trigger that makes the evidence tables append-only.** No code path in MURAQIB, however defective or malicious, can delete or alter a sealed record.

The tables are owned by the database superuser role, which **can** disable or drop that trigger. A person holding database-administrator credentials on the host is therefore not constrained by the mechanism in the way the application is.

The honest formulation: **tamper-evidence binds the application absolutely, and constrains a database administrator by detection rather than prevention.** Detection depends on the anchor. Between anchors, wholesale substitution is detectable only by comparing against the last signature.

6.2 A chain cannot prove it has been shown in full

A recipient of an evidence bundle can prove that the records it contains are internally consistent, correctly linked, and signed by the published key. They cannot prove that they were given every record.

Withholding is therefore possible, and no cryptographic property in the system prevents it. What the system offers against this is that the chain is continuous: a bundle with records missing from the middle fails verification, because the links do not reconstruct. **A complete-looking bundle that has been truncated at the end, however, verifies.**

6.3 There are no per-tenant inclusion proofs

There is **one evidence chain per environment**, not one per tenant. A tenant cannot independently prove that its own records belong to the chain without receiving the whole chain — because verification requires reconstructing every link.

This is a consequence of the chain design rather than an oversight, and it is the reason bundle export is currently an operator-mediated act rather than a self-service one (Section 10).

6.4 The evidence proves declaration and decision, not conduct

Restated here because it is the limit most easily forgotten between Section 1 and an audit: an approval records that a *declared* action was permitted. Where the declaration and the conduct diverge, the evidence follows the declaration.

6.5 Independent verification is not yet independently obtainable

The verification tool is public and requires nothing from OQIRON. **The bundle it verifies is produced by an operator.** Section 10 states this in full.

---

7. Roles and access

7.1 The role model

MURAQIB implements eight roles. Each carries a fixed set of permissions and a fixed set of dashboard views.

On the `Approve` column. It records the role's `can_approve` flag, which is returned to the browser at sign-in and is enforced on no route. It does not relate to the four-eyes control of §6.4: that control is evaluated over identities the calling system asserts and never consults a MURAQIB user role. Read this column as a dashboard capability flag, not as authority over a governed decision. The same caveat applies to `Freeze agent`, `Change mode` and `Export`; only `Manage users` and `View all evidence` gate a route.

Role	Manage users	Export	Approve	Freeze agent	Change mode	View all evidence	Read-only	Views
SUPER_ADMIN	✓	✓	✓	✓	✓	✓		30
ADMIN	✓	✓	✓	✓		✓		30
CCO		✓	✓	✓		✓		25
ANALYST		✓				✓		23
COMPLIANCE		✓				✓		19
REGULATOR		✓				✓	✓	8
EXECUTIVE		✓					✓	6
AUDITOR		✓				✓	✓	6

Three roles are structurally read-only: REGULATOR, EXECUTIVE and AUDITOR cannot approve, freeze, or alter configuration under any circumstance. The REGULATOR role exists so that a supervisory authority can be given standing access to evidence without being given the ability to affect anything.

Permission resolution deliberately avoids a fail-open default. An unrecognised role resolves to no permissions rather than to a baseline set — an important property, because the alternative would grant an unknown role the permissions of the least-privileged known one.

7.2 The reference deployment's position

Segregation of duties is a deployment control that the operating institution configures. MURAQIB provides the separations described above; whether they are in force depends on how accounts are provisioned.

At the reference deployment, **three accounts are active.** One holds every privilege; the other two hold scoped roles used to demonstrate tenant-scoped read enforcement. The remaining five accounts exist and are suspended. Segregation of duties is therefore designed and available but **not operative at the reference deployment**.

An institution deploying MURAQIB should treat account provisioning as part of its control design, and should not infer from this manual that separation is enforced by the product independently of how it is configured.

7.3 Which acts leave an evidence record — and which do not

This section states a limitation that an institution's audit function should understand before relying on the evidence chain for administrative oversight.

Nine event types exist in the chain, emitted from eight call sites:

Event	Meaning	Count at the reference deployment
`EVIDENCE_CREATED`	An agent action was declared and decided	447
`CREDENTIAL_REVOKED`	An API credential was withdrawn, with the operator and the reason	27
`CREDENTIAL_ISSUED`	An API credential was granted, with the minter and the scopes	2
`AGENT_REGISTERED`	An agent was admitted to the rail, with its environment, authority and the human who admitted it	3
`REGISTRY_ATTESTED`	An operator attested the registry's contents as at a stated moment, with a digest of each row	1
`PRIVILEGE_NARROWED` / `PRIVILEGE_RESTORED`	An agent's privileges were automatically constrained or released following behavioural deviation	1 each
`POLICY_CHANGED`	The policy core's digest changed	2
`DECISION_RESOLVED`	A human approved or rejected a held decision	1
`ENROLMENT_APPROVED`	An operator admitted an applicant to the rail — the decision, sealed before the state changed	1
`ENROLMENT_DECLINED`	An operator refused an applicant, with the reason they gave	1

Two further types are new since 15 September 2026, and one of them is a capability this manual did not previously have. Admitting an agent to the rail now leaves a chain record 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 — the issuance ordering, for the same reason. There is deliberately no HTTP route: admission is a decision a named human makes, so the surface is a reviewable command-line tool, and the self-serve provisioning path calls that same tool rather than writing the registry itself.

`REGISTRY_ATTESTED` is not a registration. The fifteen agents admitted before this mechanism existed carry no `AGENT_REGISTERED` event and **were not backfilled**: for ten of them the recorded sponsor is a machine label rather than a person, so a backdated registration would assert an accountable admission that never took place. The attestation instead records, dated the day it was made, what the registry held that day, with a SHA-256 over fourteen governing fields per row. An institution can therefore detect a later change to any attested agent's authority rather than merely observing that its admission is undocumented.

The asymmetry an auditor must not miss: admission is sealed and withdrawal is not. Freezing, suspending or revoking an agent leaves no chain record at all. Four agents at the reference deployment stand at status `revoked` with nothing in the chain naming who retired them, when, or why. §11 item 30 records this.

Two of these are new since 12 September 2026 and change what this section says. Credential revocation and credential issuance were both unevidenced when this section was first written, and the paragraph below listed issuance among the acts that leave no record. Both are now sealed: revocation records the credential identifier, the agent, the scopes withdrawn, the operator and a mandatory reason; issuance records the identifier, the agent, the scopes, the expiry and the authenticated minter. Neither records the credential itself, only the first twelve hexadecimal characters of its hash.

The two differ in one respect an auditor should know. **Revocation seals last and cannot be blocked by a sealing failure** — a credential dying matters more than the record of it dying. **Issuance seals first and a sealing failure prevents the credential existing at all.** So an unsealed live credential cannot be created, while a sealed issuance that did not complete can be, and the chain may therefore contain a grant for a credential that never reached either store.

The credentials in force before 12 September 2026 carry no issuance event and were not backfilled: appending a record today for a grant made in June or early September would place a false position in a chain whose ordering is part of its integrity claim. Their creation dates are held on the key record, not in the chain.

Everything else administrative is unevidenced. The following acts leave no record in the tamper-evident chain:

- creating a user account
- changing a user's role
- suspending or reactivating an account
- amending an agent's registration after admission (registering one is now sealed, see above)
- logging in, and failing to log in
- exporting evidence

The accurate claim, and the one this manual makes: **agent decisions are sealed, the human resolution of a held decision is sealed, the granting and withdrawal of an API credential are sealed, and the admission of an agent to the rail is sealed; the rest of platform administration — including the withdrawal of an agent's authority — is not.**

A resolution records who resolved it, on MURAQIB's own authenticated session, and carries `submitter_established` as an explicit value. On every resolution sealed to date that value is `false`: the party who caused the escalation was named by the calling system and never authenticated by MURAQIB, and `actor_id` is not a member of the canonical signed body, so it is not covered by a signature even where one is present. The event therefore establishes who resolved a decision, not who requested it, and not that the two are different people.

One refinement follows from §5.0. An agent decision is sealed **and** separately written, in the same transaction, to the unsealed operational store — and the unsealed copy is the one an operator sees on screen and receives in a spreadsheet export. The disposition is therefore threefold rather than twofold: agent decisions are sealed and separately displayed unsealed; administrative acts are neither sealed nor, in most cases, recorded at all.

An institution should not read the evidence chain as a complete record of everything that happened to the system. It is a complete record of agent decisions. Administrative actions are subject to conventional application logging and to the operating institution's own controls over server access.

7.4 Credential properties

The properties of API credentials below are stated here because they bear on an institution's key-management assessment.

Credentials do not expire. There is no expiry field in the credential store. A credential remains valid until it is explicitly deactivated.

There is no mechanism by which a production credential can be deactivated. This qualifies the sentence above and an institution should read the two together. A revocation function exists, but its only caller is a host-side tool for sandbox credentials which refuses, by explicit guard, to act on any agent that is not registered as a development agent. No administrative interface revokes. Withdrawing a production credential therefore requires a privileged database session on the host, performed by an operator; it is not an available function of the platform. Revocation is also agent-granular rather than credential-granular: one compromised credential cannot be withdrawn without withdrawing every other credential its agent holds.

A revocation is not recorded. The credential's active flag is changed in place. There is no revocation timestamp, no record of who performed it, and no chain event — so after the fact the system holds no evidence that a revocation occurred or who authorised it. An institution requiring an auditable credential-withdrawal trail must maintain it outside MURAQIB.

Scope is enforced but cannot be requested. The scopes recorded on a credential are checked, fail-closed, at the clearance routes and at chain verification. No issuance interface accepts a scope, so every credential issued today carries the clearance and read pair. Eight of the twelve credentials currently in force carry narrower scopes than that, dating from earlier provisioning; re-issuing any of them would widen it. An institution should not assume a re-issued credential matches the authority of the one it replaces.

Scope is not narrowable at issuance. Credential generation grants the clearance and read scopes together. A credential that needs only clearance cannot be issued without read.

Issuing a credential does not deactivate a prior one. An agent may hold multiple active credentials simultaneously; issuance is additive.

Issuance is not sealed (Section 7.3). The credential record carries a creation timestamp, and that timestamp is the only evidence of when it was issued.

A credential is rate-limited. Clearance calls are limited to **5 requests per second sustained, with a burst allowance of 60**, enforced per credential at the reverse proxy before a request reaches the application. Excess requests receive `429`. A caller that exhausts its burst is throttled to the sustained rate; it is not disconnected and its credential is not suspended. The limit is per credential, so two credentials receive two allowances.

Measured 5 September 2026 against the sandbox environment from a single external client: at one concurrent connection, 109 of 1,544 requests in ten seconds were served and the remainder returned `429` — consistent with a 60-request burst followed by 5 per second. At two, four, eight and sixteen concurrent connections the served count settled at 51–75 per ten seconds, the sustained rate.

The application's own concurrency ceiling is unmeasured. The rate limit binds first, so the figure above describes the limit and not the capacity behind it. During the measurement the host's load average peaked at 0.79 and the application refused nothing. **5 requests per second is a policy limit and must not be read as the system's maximum throughput**, which remains unquantified (§10.2).

The limit applies to the clearance endpoint, not to the rail. Six locations carry a rate limit. The credential-keyed limit above applies to `POST /api/v2/intercept` alone. Login, token exchange, enrolment and the public read surface carry their own limits, keyed on **client address** rather than credential. The remaining authenticated routes are served by a catch-all location with **no rate limit** — they are protected by authentication, not by throttling.

The credential key was confirmed by test on 11 September 2026: two hundred parallel requests bearing one key exhausted that key's allowance and received `429`, while requests from the same address bearing a different key were served normally. Header capitalisation does not create a second allowance — `X-API-Key` and `x-api-key` resolve to the same key and share one budget.

One consequence of the credential key is worth stating. A caller presenting neither an API-key header nor a bearer token resolves to an empty key, so **all such callers share a single allowance**. Those requests are refused by authentication regardless, but the limit does not isolate them from one another.

---

8. Tenancy

8.1 The write path

Every decision record carries the tenant that the acting agent belongs to. The tenant is **derived server-side from the agent's registration** and is never taken from the caller's request. It is sealed with the record.

At the reference deployment, 179 of 453 decision records carry a tenant identifier. One further record carries the `__TENANT_LOOKUP_ERROR__` sentinel, which records a database fault at attribution time and is not an attribution. Records created before tenant attribution was introduced do not.

8.2 The read path — in force

Tenant-scoped read enforcement is in force at the reference deployment. The configuration remains disabled by default; it is enabled here, and every serving worker was confirmed to carry the enabled value before this manual was reissued.

A reader's scope is resolved from a grant table on every request and is cached nowhere. A principal holding a grant over one tenant sees that tenant's records and no others. The enforcement is applied in the query rather than in the presentation layer, and it is applied to retrieval by identifier as well as to listing: a record outside the reader's grant is refused when requested directly, not merely omitted from a list. A principal with no grant sees nothing, which is the failure direction chosen deliberately.

What the reader is told about what they cannot see. Each scoped response carries an envelope stating the grant, the tenants in scope, the number of records visible, and whether the view is complete or partial. It does not state how many records were withheld. A reader therefore knows their view is bounded and cannot infer the size of what lies outside it.

Three limits remain, and they are the reason this section does not claim more. Two routes that read evidence apply no tenant predicate; both are presently non-functional for an unrelated defect, so they disclose nothing today, but neither is protected by the mechanism described above. An operator-level grant exists that spans every tenant, and creating a new tenant silently widens it without recording a grant row. And records written before tenant attribution existed carry no tenant at all: they belong to no grant and are visible only to the operator level.

8.3 The relationship between tenancy and verifiability

Section 6.3 states that there are no per-tenant inclusion proofs. That limitation and this one are connected, and an institution assessing MURAQIB should understand the connection.

Verification requires reconstructing every link in the chain, so a verifiable bundle must contain the complete chain. Tenant isolation requires that no customer sees another's records. **These two requirements pull against each other**, and resolving them is an architectural question rather than a configuration change. It is the subject of active work and is disclosed here rather than deferred.

---

9. Operations

9.1 Scheduled operations

The reference deployment runs six scheduled operations:

Operation	Frequency
Health and availability monitor	every 5 minutes
Anchor staleness check and staging	daily
Database backup	daily
Off-host backup replication	daily
Evidence bundle drill	daily
Backup restore drill	weekly

Two of these are worth noting specifically, because each exists to stop a claim from going untested.

The **restore drill** exercises the backups: a backup that has never been restored is a hypothesis.

The **evidence bundle drill** exercises §10. It produces a real evidence bundle from the live chain and verifies it with the independently published tool, then deliberately corrupts that bundle five ways — an altered payload value, an altered seal, a record removed from the middle, a damaged countersignature, and a reordered array — and requires the tool to reject every one. A verifier that is only ever run against good input proves that it runs, not that it detects. It also refuses to proceed if the vendored copy of the published tool does not match its recorded digest, because a verifier we could edit would not be independent.

It was added on 17 September 2026 for a specific reason, stated in §10.2: the bundle had been unproducible for eleven days and nothing detected it, because every check this system had asked the database a question and none asked the artefact to exist.

9.2 The anchor ceremony

The anchor ceremony is documented in a runbook and carries a genuine two-party separation (Section 5.5). The system stages; it never signs. The operator signs offline on a separate machine, holding the key and passphrase alone. A guarded insert is presented for approval before execution.

This is the only operational procedure for which a documented runbook exists.

9.3 Procedures that are not documented

Stated as a named gap, because an institution's operational-risk assessment will look for these:

- credential issuance
- agent onboarding
- user provisioning and deprovisioning
- incident response
- breach notification

No runbook exists for any of the above at the reference deployment. Documenting them is scoped work, not completed work.

9.4 Deployment architecture

The reference deployment runs on a **single host** with no high-availability configuration and no failover. There is no second production instance.

The availability consequence is stated in Section 3.4: an outage of that host blocks every governed agent action. This is correct fail-closed behaviour and it is also a single point of failure.

A second production instance has never been built. The reproducible-deployment work — an installable application, a reproducible chain and signing ceremony, reproducible backup and monitoring, and a proven restore into a new environment — is scoped and not yet exercised. An institution planning a deployment should treat this as a joint engineering exercise rather than an installation.

---

10. Verification

10.1 The tool

`muraqib-verify` is published under the MIT licence at `github.com/oqiron/muraqib-verify`. It reads one file and makes no network call. It does not contact any OQIRON system, requires no installation beyond a Python interpreter, and self-tests before it runs.

Its most important property is one of construction rather than function. The hashing rules are **re-implemented from the written specification carried inside the bundle**, not imported from the system that produced it. The reasoning is stated in the tool itself: a verifier that shares code with the system it verifies proves nothing, because a defect or a deliberate change in the shared code would be invisible to both sides simultaneously.

The tool checks that records are correctly linked, that seals reconstruct, and — when supplied with the public key from an independent source — that the anchor signature is valid. The public key should be obtained from a source other than OQIRON's own systems; the tool's documentation is explicit that this is the form of the check that matters.

10.2 What is independent and what is not

Confirming is independent. Obtaining is not.

A third party who has been given an evidence bundle can verify it completely without OQIRON's cooperation, consent or knowledge. That property is real and is the basis of the trust model.

A third party cannot obtain a bundle without OQIRON. Bundle generation is an operator action performed on the host. There is no self-service export path, authenticated or otherwise. In practice this means an institution requests a bundle and is sent one.

No bundle has ever been issued. No institution, regulator or auditor has received one, and until 17 September 2026 this deployment had never produced one at all. That is a harder statement than the paragraph above it and is made deliberately: §10.1 describes a verification tool that works and a format that is fully specified, and both were true of a file that had never existed outside a test.

Worse, and stated because it is the more useful fact: **from 12 to 17 September 2026 the exporter refused to produce any bundle whatsoever, and nothing detected it for eleven days.** Two independent causes, both introduced on the same day by ordinary use. A safety assertion described the payloads as containing only strings and nulls — a census of what the data happened to look like rather than a rule — and became false the moment credential events began sealing lists of scopes. Separately, the

exporter required the chain to reconstruct to the *newest* event's seal, which can only match when the newest event has already been countersigned; anchoring is periodic by design (§5.6), so the condition was almost never satisfiable. **The seals themselves verified perfectly throughout — 493 of 493 — and the evidence was never at risk.** What failed was the ability to hand the evidence to someone else, which is the entire purpose of §10. Both causes were corrected on 17 September 2026; the assertion now enforces a published bound instead of describing the corpus, and the comparison is made at the sequence the signature actually covers.

The reason eleven days passed is worth stating plainly, because it generalises: **every automated check this system had asked the database a question, and none asked the artefact to exist.** A bundle is a file. Nothing had ever produced one. An operation that is never exercised is not a capability, and a defect in it is invisible by construction. That gap is now closed by a scheduled check that produces a real bundle and verifies it with the independently published tool (§9.1).

This manual states the distinction because the alternative — describing both as independent — would overstate the position. Section 6.2 states the related limit: a recipient cannot prove the bundle they received is complete.

The load measurement of 5 September 2026 has four limits, stated so its figure is not read as more than it is.

It was generated from **one client**, whose processor reached full utilisation while producing rejected requests — the client saturated before the rail did, so the measurement cannot bound the rail's capacity.

It ran against the **sandbox** evidence chain, which uses a separate append lock from the production chain and had **no concurrent writers**. Production under real contention could be slower.

It exercised **one credential**, so it measures what one caller receives, not what the system sustains across many.

The **application's concurrency ceiling was never reached** and is therefore stated nowhere in this manual.

10.3 Published advisory

OQIRON published `ADVISORY-2026-001` for a defect that was found internally and reported by no one. It is published alongside the verification tool.

---

11. Consolidated limitations

Every limitation stated in this manual, collected for reference. **Thirty-seven items.** Each appears in full beside the capability it qualifies.

On what the system does

1. There is no enforcement point. MURAQIB returns a verdict; the calling agent honours it (§1.2).

2. The rail governs the declaration, not the payload. An approval concerns what was declared, not what was done (§1.2, §6.4).
3. MURAQIB is not an accredited certifier. Sealing is a cryptographic property, not a regulatory attestation (§1.2).

On the policy core

4. Policy provenance begins 8 August 2026. `prev_digest` is null and no earlier change is evidenced (§4.5).
5. The first recorded policy change is retroactive — the mechanism was installed after the change it records (§4.5).
6. `policy_digest` appears on 13 of 446 decision records. Earlier decisions cannot be tied to a policy version (§4.5).
7. The provenance mechanism records; it does not authorise, and it does not prevent a person with write access from changing policy (§4.4).
8. An unsigned policy change is recorded and alerted, and is permitted to serve. Strict refusal exists and is disabled (§4.4).
24. The strict setting that refuses an unsigned policy change is not effective. It blocks a single start and does not survive the automatic restart (§4.4).
25. There is no steady-state assertion that the deployed policy is signed. The signature check is reachable only on a start where the digest changed (§4.4).

On the evidence

9. Tamper-evident, not tamper-proof. The application cannot alter records; the database owner can, and is constrained by detection rather than prevention (§6.1).
10. Between anchors, wholesale substitution is detectable only against the last signature (§5.6, §6.1).
11. A chain cannot prove it has been shown in full. A bundle truncated at the end verifies (§6.2).
12. There are no per-tenant inclusion proofs. One chain per environment (§6.3).
13. Anchoring is periodic and operator-driven, not immediate (§5.6).
26. Field coverage varies across the chain. Records seal at 38 fields from 16 September 2026; the most recent existing record carries 34 and the oldest carry 25. Sealed records cannot be backfilled (§5.1).

35. **The reasoning behind every decision sealed before 16 September 2026 is permanently unsealed.** On that date the rail began sealing a machine-readable trace of the signals examined and the rules that fired, together with a SHA-256 digest of the complete trace. The 453 decision records sealed before it — every governed decision the reference deployment has taken to date, including the 230 escalations — carry no such trace and never will: the chain is append-only and enforced so by a database trigger that blocks amendment even for the database owner. Those decisions still hold a written explanation in the operational store, and that store is mutable, so the explanation of a pre-cutover decision rests on operational controls rather than on the seal. The interface states this on each such record rather than showing a blank. Two further properties of the mechanism are disclosed rather than discovered: the sealed trace is **truncated at 4 KB** where the full trace is longer, because one trace in the reference deployment runs to 645 KB and sealing every trace whole would grow the chain's

payloads roughly ninefold and rehash that value on every verification — the digest is taken over the **complete** trace before truncation, so a full trace presented later can be verified against the seal and a truncated record announces itself; and the **rendered prose explanations are deliberately not sealed**, because they are generated from the decision, the rule fired and the risk score, all four of which are already sealed. Sealing a rendering of sealed fields would prove nothing the record does not already prove, and would invite a reader to treat generated text as a finding (§5.1).

31. **Human resolution of a held decision is sealed; the operational store's approval columns are a cache of it, and a lossy one.** Four-eyes enforcement exists and is real, but it operates on the clearance path: the rail evaluates authoriser role and maker-distinctness over fields the calling system asserts (§2.1 stage 4, §6.4). A person resolves a held action through `/api/escalations/<evidence_id>/resolve`, which requires an authenticated MURAQIB session, refuses a body-supplied approver outright rather than ignoring it, and seals `DECISION_RESOLVED` in the same transaction as the operational write — so a resolution that was returned is a resolution that was sealed (§3.2, §7.3). **What is sealed does not establish who requested the action.** The event carries `submitter_established="false"` and states in terms that MURAQIB did not authenticate the party who caused the escalation: `actor_id` is not a member of the signed canonical field set, so it is not covered by a signature even when one is present. The event establishes who resolved the escalation and when, on MURAQIB's own session; it does not establish that resolver and requester are different people.

36. **Enrolment decisions taken before 18 September 2026 are not sealed, and one of them is recorded nowhere but a mutable table.** From that date the decision to admit or refuse an applicant seals before the state changes, and a seal failure leaves the request in the queue rather than granting access (§7.3). Two decisions predate the mechanism. The **approval** of 4 September 2026 is covered indirectly and verifiably: `REGISTRY_ATTESTED` at chain sequence 521 carries a digest of the agent it produced, and the registry row inside that digest records `registered_at 2026-09-04` and `registered_by "self-serve signup #90"`. That is an attestation of state as at 15 September, which is what it is — not a record of the decision, but a sealed statement that the thing the decision created exists and has these properties. The **refusal** of 4 September 2026 is covered by nothing and cannot be. A decline creates no registry row, so attestation does not reach it; it exists only in a mutable table, it carries no recorded reason because the field was never written before this date, and the person who took it is identified by a username rather than an account id. Sealing it now would place a fabricated date in an append-only record, so it is disclosed instead. This is the strongest argument for the reason field being mandatory from now on: an approval leaves a tenant, an agent and a credential, all sealed downstream, whereas a refusal leaves only what was written down at the time.

37. **Two certificate-shaped views were removed from the interface on 18 September 2026.** The first rendered an ISO/IEC 42001 conformity status, a certificate identifier, an issuing authority, an expiry date and a score across ten named domains, derived from counts in the evidence store. It was removed rather than corrected, because two of its domains asserted conformity in areas this product does not address: one reported *Training Data & Model Governance* as fully satisfied on a rail that governs no training data, and another reported *Human Oversight & Intervention* as fully satisfied on the same system this manual discloses as establishing no submitter identity for a human resolution (item 31). Two domains had no negative branch at all and so could not fail on any data, including an empty database; two others were identical expressions and could never disagree; and every threshold

deciding them was a literal in application code, outside the signed policy core, editable with no ceremony, no digest change and no `POLICY_CHANGED` event — while none of the 58 controls in that core can be altered without all three.

The second rendered per-agent **governance passports** with certificate identifiers and ISO 42001, PDPL and SAMA compliance flags. Those flags were constants in application code rather than findings, and **the five agents they described were not present in the agent registry** — they were compliance assertions, carrying certificate identifiers, trust scores, issue and expiry dates and financial authority limits, about agents that did not exist on this rail. Removing the first view while leaving the second would have moved the claim rather than withdrawn it. A third instance was removed in the same movement: the evidence-record detail panel rendered a risk tier and a certification status as fixed strings on **every** record, regardless of agent, action or verdict.

No certificate identifier from any of them was ever published, exported or quoted outside the interface. Neither the evidence bundle nor the spreadsheet export carried one, and no identifier appears in this dossier, on oqiron.ai or in the SDK. OQIRON is not certified to ISO/IEC 42001, PDPL or SAMA and does not claim to be; the public statement to that effect is unchanged, and these views were the only place the product said otherwise. The measured figures beneath the first view — actions governed, blocked, escalated, risk-scored and sealed, active policies, policy versions and distinct agents — remain available as an activity summary, which is what they always were. A check asserts that the removed routes continue to return 404 and that no withdrawn claim reappears in the interface; that check tests route absence and a fixed list of field names, so a differently-named reintroduction would not be caught by it.

The columns are derived, not authoritative. For every sealed escalation the resolution state is recoverable from the chain alone — take each `EVIDENCE_CREATED` whose decision is `ESCALATED` and look for a `DECISION_RESOLVED` naming it — and that derivation agrees with the operational columns exactly: as at 17 September 2026, 229 unresolved and 1 resolved, with no row disagreeing. The chain record carries fourteen fields where the operational store has three, so the columns hold strictly less than the chain and nothing that the chain does not.

The cache cannot represent a rejection. `approved_by` and `approved_at` are written for both outcomes; only the sealed event carries `outcome`. A rejected resolution therefore writes the resolver's identifier into a column named `approved_by`, and nothing in the operational store distinguishes it from an approval. No rejection has yet been recorded, so this is a latent defect rather than a present misstatement — but an institution reading the operational store must not treat a populated `approved_by` as evidence of approval. The sealed event is the record; read `outcome` there. The two columns were removed from the spreadsheet export on 17 September 2026 for this reason, and the interface labels the field "marked reviewed by" rather than "approved by".

Three rows carry a value, and they are three different things. `approved_by` is populated on 3 of 1,073 operational rows. One (`EVD-BFC347CC`, resolver `usr_001`) is a genuine resolution, sealed at chain sequence 481 and derivable as above. The other two (`MRQ-2026-000001`, `MRQ-2026-000005`) predate sealing entirely — they carry no chain record of any kind, were written under the previous SQLite-era store by a function that has no caller today and writes to a database this system no longer uses, and they name individuals in free text who hold no account. **Those two are permanently unattributable**: no session established the identity, no chain entry fixes the time, and they cannot be derived because there is nothing

sealed to derive them from (§7.1, §7.3). **Approval of an enrolment request is a separate matter and is not sealed at all** — no chain event type exists for it, so unlike a held-decision resolution it is not derivable (§7.3).

30. Agent lifecycle has no application surface, and the containment store fails open. **Admission was separated from the rest of the lifecycle on 15 September 2026 and is the one act that now leaves a chain event** (`AGENT_REGISTERED`, §7.3), sealed before the registry row is created and performed through a reviewable command-line tool. That there is no HTTP route for it is now a deliberate position — admitting an actor is a decision a named human makes — rather than an omission. **The remainder of the lifecycle is unchanged and is the sharper half.** Freezing, suspending or revoking an agent are still performed by an operator with direct database access; no HTTP route exists for any of them, `agent_control` grants the application SELECT only, and no write to it appears anywhere in the codebase. An agent cannot alter its own containment state. Three consequences follow: **withdrawal of an agent's authority leaves no chain event**, so the register now seals at one end and not the other, and four agents at the reference deployment stand at status `revoked` with nothing naming who retired them, when, or why; the mechanism behind the remaining disclosure at §7.3 is that the routes do not exist rather than that they omit sealing; and `can_freeze_agents` appears in the role model, in §7.1's table and in the login response but is enforced on no route, describing a capability the application does not have. Separately, the freeze lookup fails open — if the control store is unreadable, the state resolves to not-frozen rather than manufacturing a freeze, on the stated grounds that a containment control must not block every action on a transient database fault. The kill-switch is therefore defeated by the store's availability rather than by its subject, and no alert distinguishes "not frozen" from "could not determine" (§7.3).
29. Content is parsed on one path. The clearance path never receives the substance of an action, but the declared-summary scanner receives and parses proposed output text in order to detect personal data. It stores derived metadata only; the scan log is outside the evidence chain, has no immutability trigger, and the application holds delete on it (§5.2).
28. The evidence surface and the evidence chain are different stores. What the interface and the spreadsheet exports display is the unsealed operational store; what the verification tool checks is the sealed chain. They are joined by evidence identifier and written atomically, but 620 of the operational store's 1,073 rows have no sealed counterpart (§5.0; whitepaper §11.2.20). **From 17 September 2026 both export artefacts state on their face which store they were drawn from**, and name the independent verification tool for the other one — a recipient no longer has to know this limitation exists in order to read the file correctly.

On roles and administration

14. Segregation of duties is designed and configurable; at the reference deployment three accounts are active, one holding all privileges (§7.2).
15. Agent decisions are sealed, the human resolution of a held decision is sealed, the granting and withdrawal of an API credential are sealed, the admission of an agent to the rail is sealed, and **from 18 September 2026 the decision to admit or refuse an applicant is sealed** — both outcomes, the refusal carrying the reason given (§7.3). The rest of platform administration — user creation, role change, suspension, login, amendment of an existing agent, **withdrawal of an agent's authority**, evidence export — is not (§7.3).

16. Credentials do not expire, cannot be scope-narrowed at issuance, and are additive rather than replacing (§7.4).

On tenancy

17. Tenant-scoped read enforcement is in force. Two evidence-reading routes apply no tenant predicate, an operator grant spans every tenant and widens silently when a tenant is created, and pre-attribution records belong to no grant (§8.2).
18. 179 of 453 decision records carry a tenant identifier. One further record carries the `__TENANT_LOOKUP_ERROR__` sentinel, which records a database fault at attribution time and is not an attribution (§8.1).
19. Verifiability and tenant isolation pull against each other; the resolution is architectural and in progress (§8.3).

On operations

20. No runbook exists for credential issuance, agent onboarding, user provisioning, incident response or breach notification (§9.3).
21. Single host, no failover. An outage blocks every governed action (§3.4, §9.4).
22. A second production instance has never been built (§9.4).
27. Signature-pin durability is partial and manual: the pin for the deployed digest has been copied to durable storage, but nothing schedules that copy and neither location is replicated off the host, so a cold start taken before a copy reports a signed policy as unsigned. No control-level manifest exists for the deployed digest anywhere, so the next recorded policy change cannot state what changed (§4.5; whitepaper §11.5.13b).

On verification

23. Confirming a bundle is independent of OQIRON. Obtaining one is not (§10.2; whitepaper §11.2.19).
33. The clearance path's concurrency ceiling is unmeasured. A per-credential rate limit of 5 requests per second binds before the application does, so its maximum sustained throughput has never been established. The published figure is a policy limit, not a capacity (§7.4; whitepaper §11.2.29).
34. The load measurement of 5 September 2026 was made from a single client, against the sandbox chain with no concurrent writers, using one credential, and the client saturated before the rail did (§10.2).
32. Approval provenance diverges from approval record. A resolution is sealed with the resolver's session-derived identity, but the party who caused the escalation is carried as a client-supplied value outside the signed field set. Every resolution sealed to date records maker–checker as unevaluated (§7.3; whitepaper §11.2.21).

This manual is one of five documents in the OQIRON trust dossier. D1 is the verification tool; D2 the security whitepaper and its limitations register; D3 the PDPL and NCA-ECC alignment mapping; D4 the OQIRON Platform Manual, a strategic overview for a non-technical reader; D5 this manual.