Provably Compliant Autonomous Actions: A Consensus Enforced, Dual Authority Certification Gate with Freshness Bound Per Action Proofs

Jason Duke (Kronaxis Limited)

7 June 2026

Abstract

Autonomous artificial intelligence agents and automated pipelines increasingly take real world actions in regulated industries: outbound calls, written communications, transactional commitments, regulated data disclosures. The operating organisation must be able to prove, after the fact and to a regulator or counterparty, that each such action conformed to the applicable rules. Existing answers either trust the operator (centralised audit logs, application layer policy gateways) or fail to bind the proof to the action at consensus time (single authority on chain attestations, transparency logs without policy co residency). We present a method by which an autonomous action is made structurally incapable of execution unless, at the moment of admission and on a distributed ledger substrate, it carries valid role typed attestations from a regulator aligned certifier and from the tenant's own controlled certifier, against a deterministically composed ruleset whose version is currently active for that tenant, within an open validity window. The consensus rule of the ledger refuses to admit any action lacking these bindings. We define six gate invariants (three decision time: non bypass, hash binding, composed policy admissibility; three holding over time: refusal code completeness and soundness, consensus time non admit, and freshness bound temporal admissibility) and demonstrate a four validator Byzantine fault tolerant reference embodiment in which the structural invariants reproduce across 26 scenarios. We evaluate adversarial robustness against three independent red teams (43 findings: 10 critical, 12 high, 13 medium, 8 low) and benchmark the augmenting large language model evaluator at 98.86% joint label accuracy and 99.16% refusal precision and recall on n=700. The freshness binding invariant closes the gap between compliant at decision and compliant at action and provides a formal answer to the practical question "verify a subject is not self excluded without a real time lookup". The paper publishes the concept; the production codebase, the on the wire protocols, the canonicalisation rule, the rule pack content, the operating numbers, and the deployment topology remain Kronaxis Limited trade secret.

Contents

1. Introduction

UK regulated industries are absorbing autonomous artificial intelligence into the outbound action surface faster than the audit infrastructure that governs that surface can keep up. Gambling operators run automated CRM journeys. Consumer credit firms run agentic collections. Direct marketing operators run AI assisted creative pipelines. Pharmacy and energy verticals run automated outreach and triage. The regulator's question on the next post incident interview is unchanged from the human era: prove that each action conformed to the applicable rules at the moment it was taken. The audit answer the operator can today give a regulator or a buyer's General Counsel is materially weaker than it was for the human era.

1.1 The regulatory proof problem

Regulated organisations must demonstrate, to a regulator, an auditor, a customer, or a court, that a specific historical action conformed to the rules that governed it at the time. When the action was taken by a human employee under an internal control framework, the proof reduces to "our records say so" plus the credibility of the framework. When the action is taken by an autonomous AI agent or by a machine driven pipeline, the same posture is no longer sufficient. The action is opaque to the buyer's General Counsel; the framework is opaque to the regulator; the record is held by the operator of the very system being audited.

Three failure modes recur across the current generation of answers:

1. **Pre deployment training and prompting.** Instructing a language model to follow the rules is not auditable, not deterministic, and not evidence. It produces no per action record. It can be defeated by adversarial inputs or by emergent behaviour, and the next model upgrade can silently change the policy surface.

2. **Centralised audit logs.** The platform that runs the agent records what it did. The tenant must trust the platform's honesty, the platform's controls, and the platform's record retention. There is no cryptographic per action proof and no mechanism by which the regulator or the tenant's counsel can independently verify, against an external source, that any given historical action did what the log claims.
3. **Application layer policy gateways.** A policy engine sits in front of the agent's outgoing actions and filters them. Better than nothing, but it remains operator trust: the platform controls the policy, the engine, the logs, the deployment, and the key material that signs any receipts produced.

A regulator or a tenant's counsel asked to rely on any of these is being asked to trust the operator's word about the operator's own behaviour.

1.2 What the method establishes

This paper consolidates and formalises a method by which an autonomous action cannot execute, at the consensus layer of a distributed ledger, unless it carries valid role typed attestations from a regulator aligned certifier and from the tenant's own controlled certifier, against a composed ruleset whose version is currently active for that tenant. The method makes three further design choices that distinguish it from prior runtime governance work:

- The active ruleset is the deterministic composition of two authority layers (regulator baseline and per tenant overlay), with both layers signed and ledger resident.
- The role typed quorum architecturally requires a tenant controlled signer. The platform operator cannot, alone or in combination with any other operator side party, activate a ruleset or admit an action against a tenant whose signer has not co signed.
- Every PERMIT is bound to the freshness of the reference data it relied on (self exclusion register, consent withdrawal, sanctions list, age verification, affordability data, hours of operation). A permit is non admissible outside the named validity window of those registers or against a superseded ruleset.

The freshness binding property is the formal answer to a practical question that every gambling, financial services, and direct marketing buyer asks: how does the system verify that a subject is not self excluded without a real time lookup against the upstream register at the moment of every action. The answer is that the permit is bound to the register snapshot version and an explicit validity window; the gate refuses any permit that is older than the contracted freshness service level agreement; an in window invalidation set is consulted at the per action send step so that a fresh self exclusion registered between pre certification and send is honoured. The check is local and fast; the freshness is named in the rule pack and stamped on the proof.

1.3 Contributions

This paper makes four contributions:

1. **A bounded and testable security model for runtime AI compliance.** The six gate invariants (three decision time, three holding over time) define what the gate does at the block boundary and what the record continues to mean as time passes. Every invariant is paired with a mechanism, a stated limit, and a reproducible test in the reference embodiment.
2. **A consensus rule binding for the certification check, not a middleware filter.** The certification check runs in every honest validator's pre execution hook at block proposal time; the Byzantine fault tolerant supermajority assumption protects the gate. No single validator and no platform operator alone can admit an uncertified action.
3. **A dual authority composed ruleset with a tenant required signer.** The operating ruleset is the deterministic composition of a regulator aligned baseline and a per tenant overlay, both signed under different authority and both ledger resident. The quorum policy architecturally requires the tenant's own controlled signer; the platform operator cannot, alone or in combination with any other operator side party, activate or admit against a tenant whose signer has not co signed.
4. **A freshness bound temporal admissibility property.** Every PERMIT is bound, by named freshness service level agreement, to the version of the upstream register the gate relied on. The "without a real time lookup" question receives a named, contracted, stamped, fail closed answer.

The paper also reports a four element evaluation corpus: 26 reproducible scenarios on a four validator Byzantine fault tolerant reference cluster; a wider 105 unit test plus 13 policy file plus 118 rule pack scenario hardening pass; three independent red teams producing 43 deduplicated and prioritised adversarial findings; and a 700 record benchmark of the augmenting large language model evaluator at 98.86% joint label accuracy.

1.4 What this paper does and does not disclose

This is a method level white paper. It discloses the conjunction of design elements, the architectural shape of the gate, the security properties the gate provides, the evaluation evidence, and the limitations and residual risks. It does not disclose the production codebase, the on the wire protocols, the exact canonicalisation rule, the per vertical rule pack content, the operating numbers, or the deployment topology. Those remain Kronaxis Limited's property and are held as trade secret. The paper publishes the concept; the product is closed.

The paper builds on, supersedes, and cross references the public facing documents in the Kronaxis Compliance documentation set: the defensive publication (Duke, 2026a), the trust and assurance white paper (Duke, 2026b), the system specification (Duke, 2026c), the threat model (Duke, 2026d), the prior art sweep (Duke, 2026e), the red and blue adjudication report (Duke, 2026f), and the AI evaluator benchmark (Duke, 2026g). Where any inconsistency would otherwise arise, this consolidated white paper is authoritative for the architecture and security property text. The codebase remains authoritative for what is built today.

1.5 Roadmap

Section 2 surveys the related work and positions the method against it honestly. Section 3 sets out the architecture: the consensus enforced gate, the dual authority composed ruleset, the role typed quorum with a mandatory client signer, the per action proof, and the egress and enforcement points. Section 4 states the six gate invariants and explains the formal answer to the freshness question. Section 5 sets out the threat model and the structural defences against named attacks. Section 6 reports the evaluation evidence from the built reference embodiment. Section 7 explains the two deployment modes (Observe and Enforce) and the retrofit pattern for an existing pipeline. Section 8 states the limitations and the honest residual: what the method does not claim, and what the trust hardening commission closes. Section 9 concludes. Section 10 acknowledges contributing parties. Section 11 lists references. Appendix A reproduces the published refusal code taxonomy. Appendix B describes the source repository layout in functional terms.

2. Related work

The method touches four neighbouring fields. We summarise the relevant prior art briefly and state where the method differs.

2.1 Deterministic policy languages

A family of deterministic, formally specifiable authorisation policy languages with bounded time evaluation and static analysis is now public art, with Open Policy Agent and its Rego language (Open Policy Agent Project, 2024) as the de facto policy decision point across cloud native systems, Cerbos (Cerbos Project, 2024) targeting the AI agent and microservice surface with policy overlays, and Oso (Oso Project, 2024) providing the Polar language for tenant aware policies. Several entries in this family also expose a static analyser capable of proving policy properties (subset, equivalence, idempotence) at compile time.

All are deterministic policy evaluators. None of them, by themselves, address the question of how a policy decision is bound to a real world action, where the policy version came from, or how a third party verifies a decision against a public substrate without the policy operator's cooperation. They are the leaf policy layer; the method extends them upward into the consensus rule layer.

The static analysis story (formal verification of policy properties) for this family of languages is particularly valuable in a multi authority composition: the composer can statically prove "the composition is at least as strict as the baseline", which prevents the silent loosening failure mode that an audit cannot otherwise detect at the rule pack layer.

2.2 Byzantine fault tolerant consensus

Several production Byzantine fault tolerant consensus engines expose deterministic block proposal hooks usable for pre execution policy checks: a pre execution callback

that runs on every validator before it pre votes on a proposed block, and a finalisation callback that runs on every validator on commit. The method uses one such engine as the reference embodiment substrate and the BFT property reproduces in the four validator test cluster (Section 6.1). The method is engine independent: any deterministic Byzantine fault tolerant consensus protocol that exposes a pre execution hook at block proposal time suffices.

Permissioned ledgers such as Hyperledger Fabric (Hyperledger Project, 2024) provide an alternative substrate. The framing in Bagheri *et al.* (2025) ("A Blockchain Monitored Agentic AI Architecture for Trusted Perception Reasoning Action Pipelines") teaches monitoring and verification of agent action against an immutable record. The method differs in that the gate is enforced as a consensus rule of the engine, not as a downstream verifier, and the ruleset authority is a typed quorum rather than a single attester.

2.3 Transparency logs and signed evidence

A family of production grade, append only, cryptographically verifiable, publicly readable transparency logs is now public art, with mature client libraries and well understood inclusion proof semantics. Certificate Transparency RFC 6962 (Laurie, 2013) and its successor RFC 9162 (Laurie *et al.*, 2021) establish the public tamper evident log primitive for TLS certificates.

The method's per action proof co resides on the same ledger as the ruleset registry, the certifier registry, and the quorum attestation registry. An external, publicly readable transparency log is the named mirror in the design, so that public verifiability is available without requiring the auditor to operate a validator (Section 7.3). The method differs from the transparency log pattern in one structural way: a transparency log records that an artefact existed at a time; the method's gate records that a policy decision was enforced at a consensus rule layer before the artefact was produced. The downstream auditor can verify both.

2.4 AI agent runtime governance

The Open Agent Passport (South *et al.*, 2026), the OpenPort Protocol (OpenPort Project, 2026), Microsoft's Agent Governance Toolkit (Microsoft, 2026), and Attested Intelligence's Attested Governance Artifacts (Attested Intelligence, 2025) all teach pre execution authorisation of agent tool calls against a declarative policy with a signed audit record. The bare pre execution authorisation pattern is now public art. The method's surviving novelty against this prior art is the conjunction of (a) dual authority composable policy, (b) role typed quorum architecturally requiring a tenant controlled signer, (c) consensus rule refusal at the dispatch layer of a Byzantine fault tolerant ledger, and (d) co resident per action proof on the same ledger as the policy and the quorum. No prior work combines these four.

2.5 Governance, risk, and compliance platforms

OneTrust (OneTrust, 2026), ServiceNow GRC (ServiceNow, 2026), Credo AI (Credo AI, 2024), Holistic AI (Holistic AI, 2024), and IBM watsonx.governance (IBM, 2024)

provide governance, lifecycle monitoring, and compliance reporting. These are documentation and lifecycle platforms; they do not enforce at the action layer and they do not produce cryptographic per action proof. The method's positioning against this category is on cryptographic verifiability and multi signer policy authority, not on workflow.

2.6 What this paper inherits and what it adds

The paper inherits the deterministic, formally specifiable policy evaluator primitive from the authorisation policy language family, the Byzantine fault tolerant consensus primitive from the deterministic BFT consensus engine family, the tamper evident append only log primitive from the public transparency log family, and the agent action interception pattern from the recent AI agent runtime governance literature. It adds the consensus rule binding, the role typed dual authority quorum, the per action proof co residency, the freshness binding temporal admissibility property, and the bounded claim discipline that holds the system to what it can actually prove rather than what it would be commercially convenient to assert.

3. Architecture

This section describes the architecture in five components: the canonical action descriptor, the dual authority composed ruleset, the role typed certifier quorum with a mandatory client signer, the consensus enforced gate, and the per action proof. The reference embodiment uses a memory safe systems programming language, a four validator deterministic Byzantine fault tolerant consensus engine with a pre execution block proposal hook, a deterministic formally specifiable authorisation policy language as the leaf policy layer, an unforgeable digital signature scheme for signatures, and a standard cryptographic hash function for fingerprints. These choices are illustrative rather than load bearing: any equivalent deterministic substrate suffices for the security properties in Section 4.

3.1 Canonical action descriptor

Every regulated action is normalised to an `ActionDescriptor`: a typed record carrying the action kind (`sendPromo`, `sendResponsibleGambling`, `sendTransactional`, `voiceCallOutbound`, `paymentInit`, and so on), the tenant identifier, the persona or operator identifier, a target (an E.164 number, a channel handle, an account identifier, a recipient address), a cryptographic hash of the canonical payload, the jurisdiction, the vertical, the contextual flags (recipient self exclusion status, vulnerability flag, affordability band, age band, hour of day, operating hours window), the consent reference, the lawful basis, the ruleset identifier the actor intends to be certified against, the validity window, and an antireplay nonce. The descriptor canonicalisation is deterministic by specification: any independent implementation that follows the canonicalisation rule produces byte identical bytes from byte identical logical content. The hash function is a standard cryptographic hash. We refer to the cryptographic hash of the canonicalised descriptor as `H(d)`.

Hash binding (Section 4.2) means every downstream artefact references `H(d)` directly. Change one byte of the payload and `H(d)` changes; the existing signatures no longer verify against the changed bytes; the existing proof no longer points at the changed action.

3.2 Dual authority composed ruleset

The operating ruleset for a tenant is the deterministic composition of two authored layers:

- A **regulator aligned baseline**: the published rules of the regulator for the vertical and the jurisdiction (United Kingdom Gambling Commission's licence conditions and codes of practice; Financial Conduct Authority's CONC handbook; Information Commissioner's direct marketing guidance; General Pharmaceutical Council's standards; Office of Gas and Electricity Markets supply licence conditions). The baseline is encoded as a set of deterministic formally specifiable policies; it is versioned by a `RegulatorBaselineVersionHash` and is ledger resident.
- A **per tenant overlay**: the tenant's own counsel's additional controls, written by the tenant's GC team, signed by the tenant's key, and deposited on the ledger as `TenantOverlayVersionHash`. The overlay is normally at least as strict as the baseline; the composer refuses to publish a composition at compile time where the overlay is intentionally more permissive than the baseline, and the deviation has to be made explicit and signed.

The composition rule is itself a named, versioned, ledger resident artefact (`ComposeRuleVersionHash`). The composed ruleset's identity is the triple (`RegulatorBaselineVersionHash, TenantOverlayVersionHash, ComposeRuleVersionHash`), which hashes to `ComposedRulesetHash`. Every action's certification binds to a specific `ComposedRulesetHash`; any third party with read access to the ledger can independently reconstruct what rules applied when.

The composition rule is conservative: a deny in either component is a deny in the composition. The overlay is at least as strict as the baseline. The leaf language's static analyser proves at compile time that the composition's permit set is a subset of the baseline's permit set; a violation at this layer is a compile time error.

3.3 Role typed certifier quorum with a mandatory client signer

A composed ruleset is not active until a quorum of role typed certifiers has co signed its `ComposedRulesetHash`. The role types are at least two:

- `RoleRegulatorBaseline`: a regulator aligned certifier whose attestation says that the regulator baseline portion of the composition is faithful to the regulator's published material. In the reference embodiment, this signer is Kronaxis operated and is named `did:kxc:regulator-baseline:kronaxis`.
- `RoleClient`: a tenant controlled certifier whose attestation says that the per tenant overlay portion is faithful to the tenant's counsel's content and that the tenant accepts the composition. The signer's tenant identifier must match the tenant identifier in any action descriptor it authorises.

An optional RolePlatformOperator and RoleHumanSigner may also be present. The quorum policy is M-of-N where M includes at least one valid signature from RoleRegulatorBaseline and at least one valid signature from RoleClient. Neither alone is sufficient; the platform operator's signatures cannot, in any number, activate the bundle alone.

Cross tenant binding is enforced at the consensus layer: a signature from tenant Alpha's RoleClient signer does not satisfy the RoleClient slot for tenant Beta. The validator's quorum check rejects any attestation whose tenant required signature does not bind to the same tenant identifier as the action descriptor under consideration, with the named refusal code CROSS_TENANT_BINDING.

The method's design contract is that the RoleClient signer's key is held by the tenant, on tenant controlled infrastructure, and is not also held by the platform operator. In the reference embodiment today the per tenant digital signature key is generated and held in software inside the platform operator side process for demonstration purposes; the swap to an out of process client hosted signer over a mutually authenticated channel is the named trust hardening row "Client hosted signer (Layer 1)" and is commissioned but not yet wired (Duke, 2026b, Section 7). The signer interface (quorum.Signer) is built and is the swap point. The architectural property holds today at the cross tenant binding layer and at the role typed quorum at consensus layer; operational removal of platform operator capability requires the Layer 1 swap and the multi party validator distribution (Section 7.2).

3.4 Consensus enforced gate

The certification check is a rule of the distributed ledger's consensus engine, not a middleware filter. Every validator node, before allowing an agent action transaction to be included in a block, independently runs the following six checks:

1. Recompute the canonical hash H(d) of the action descriptor.
2. Look up the matching quorum attestation row addressed by H(d) and ComposedRulesetHash.
3. Verify each role typed signature on that attestation against the certifier registry at the block height under consideration.
4. Verify that the recorded ComposedRulesetHash matches the currently active composition for the tenant identified in the descriptor.
5. Verify that the attestation's validity window is open at the block height under consideration and that the freshness stamps for every register the rule relied on are within their contracted service level agreement (Section 4.6).
6. Refuse to pre vote for any proposed block containing an action that fails any of the above checks; the refusal carries a code from the published taxonomy (Section 4.4).

The structural consequence is that the gate cannot be bypassed by an alternative remote procedure call endpoint of any single validator, and a single Byzantine validator within the configured fault threshold cannot cause an uncertified action to commit. The honest supermajority of validators independently refuses, and the proposed block fails to reach consensus. The reference embodiment reproduces this property under

test: scenario T5 in the Phase-3 reference implementation exercises a single Byzantine validator (within the configured `f` fault threshold) pre voting for an uncertified action; the honest three of four refuse; the block does not commit; the action counter remains zero cluster wide.

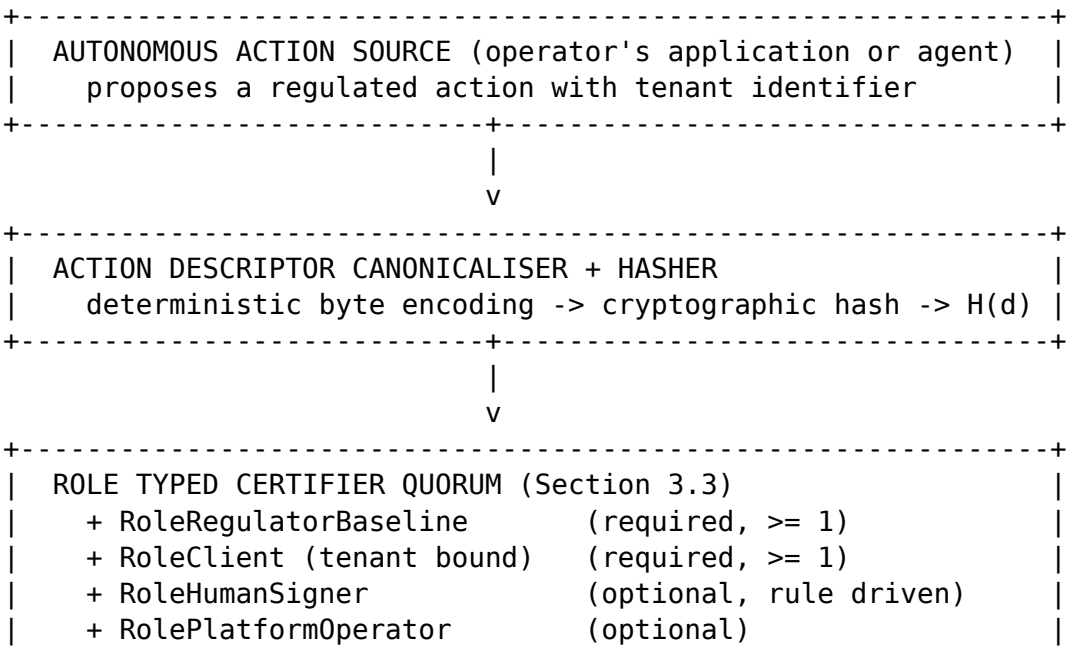
3.5 Per action proof

Each admitted action emits a per action proof record bound to the same ledger. The record binds, at minimum:

- The canonical hash `H(d)` of the action descriptor.
- The `ComposedRulesetHash` under which the action was certified.
- The identifier of the quorum attestation that authorised it.
- The action class.
- The canonical hash of the action's output.
- The block height at which the admission was decided.
- The reference data freshness stamps per consulted register (`RegisterID`, `VersionHash`, `AppliedAtUnix`, `CheckedAtUnix`, `AgeAtCheckSeconds`, `SLAWindowSeconds`).
- The tamper evident inclusion proof binding the record to the block header at which it was committed.

The record is addressed by `(ComposedRulesetHash, action_class)` so that any third party with public information about the tenant's active ruleset can locate every action of a given class committed under that ruleset. Co residency on a single ledger with the policy bundle, the certifier registry, the quorum attestation registry, and the per action proof registry is structural: it makes the proof legible to a non specialist auditor without correlating three independently administered systems.

3.6 Architecture summary



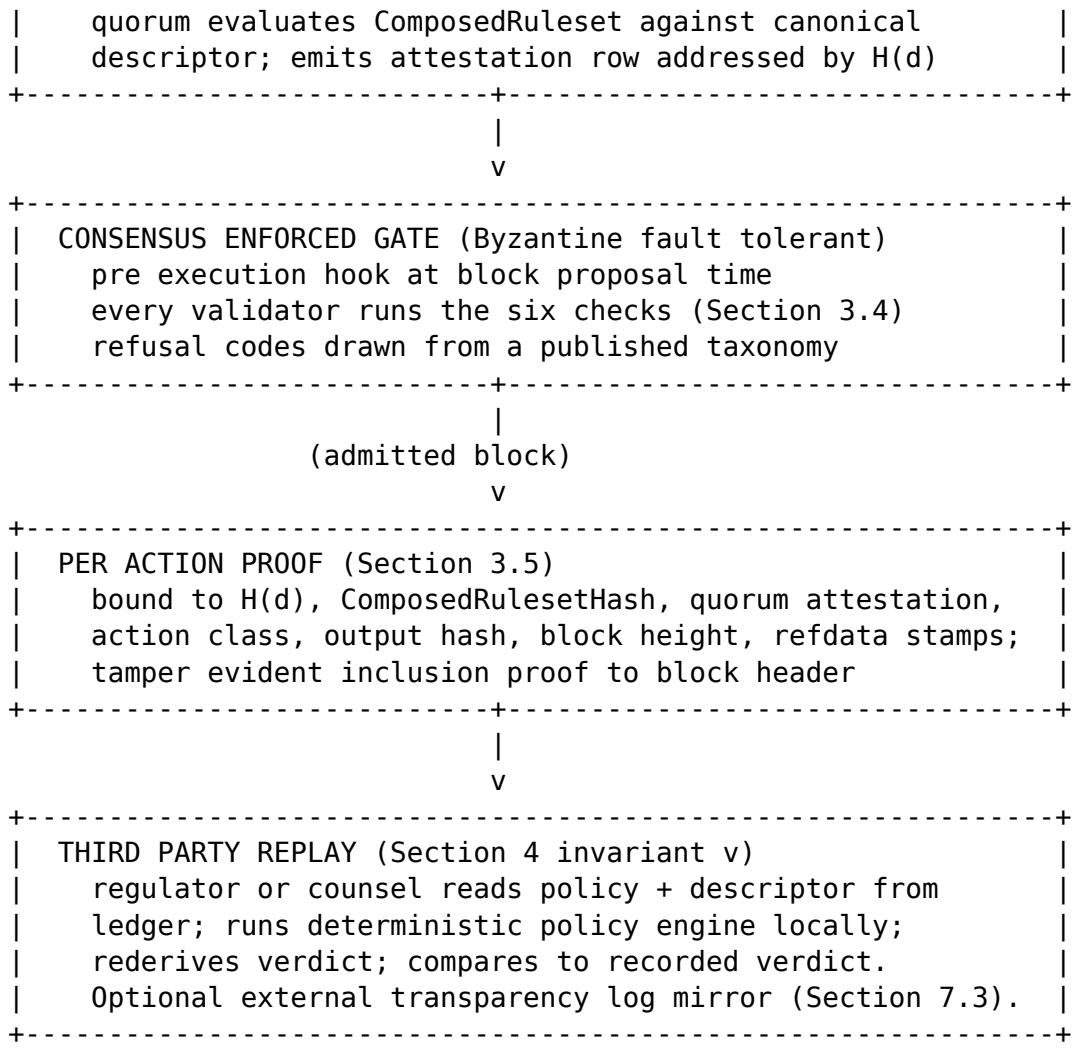


Figure 1. Architecture summary. The gate is enforced as a consensus rule at block proposal time. The third party replay path closes the loop without requiring the auditor to operate a validator.

4. Security properties

We define six gate invariants. We organise them into three decision time invariants (the properties that hold at the moment the gate decides admission) and three holding over time invariants (the properties that hold of the record after admission, across the history of the ledger). The decision time set establishes what the gate does at the block boundary. The holding over time set establishes what the record continues to mean as time passes and as the world changes around it.

4.1 Invariant (i): non bypass

Statement. No block is admitted by the honest supermajority of validators that contains an action whose canonical descriptor hash `H(d)` is not bound, at the block height

under consideration, to a quorum attestation row whose role typed signature set satisfies the active quorum policy of the active composed ruleset for the tenant identified in the descriptor.

Mechanism. Every honest validator independently runs the six checks of Section 3.4 in its pre execution hook (`ProcessProposal`) before voting. A validator that votes for a block containing a non bypass violating transaction is itself Byzantine, and is constrained by the BFT honest supermajority assumption.

Limits. Non bypass holds against a single Byzantine validator within the configured fault threshold (the reference embodiment runs `N=4`, `f=1`). It does not hold against a coalition above the fault threshold. The trust topology (Section 7.4) addresses this by distributing validators across mutually distrusting parties.

4.2 Invariant (ii): hash binding

Statement. Every certification, every quorum signature, every per action proof, and every on chain violation record references `H(d)`. Changing any byte of the canonical descriptor changes `H(d)`, invalidates the existing signatures against the changed bytes, and breaks the binding between the certification and the action.

Mechanism. The descriptor canonicalisation is deterministic by specification (length prefixed binary encoding; sorted field order; no ambient state). The hash function is a standard cryptographic hash. The signature scheme is an unforgeable digital signature scheme. The verifier recomputes `H(d)` from the descriptor on the wire and rejects any mismatch.

Consequences. The certification cannot be detached from the action it certified. The action cannot be modified after certification without breaking the certification. A verdict object cannot be swapped onto a different action; a creative cannot be modified after pre certification without the per message hash equality check failing at the egress chokepoint.

4.3 Invariant (iii): composed policy admissibility

Statement. A block is admitted by the honest supermajority only if the ruleset under which admission was decided is a composed ruleset whose `ComposedRulesetHash` is currently active for the tenant identified in the descriptor, where activation requires at least one valid signature from `RoleRegulatorBaseline` and at least one valid signature from `RoleClient` whose tenant identifier matches the descriptor.

Mechanism. The composer is a deterministic merge function whose source and version are themselves ledger resident. The leaf language's static analyser proves the composition is at least as strict as the baseline at compile time. The composed ruleset, its constituents, and its activation attestations are addressed by hash on the ledger. The validator looks up the active composition for the tenant at the block height and runs the deterministic policy decision engine over the canonical descriptor against the active composition.

Consequences. There is no single authority over the operating ruleset. There is no operator only path to change it. A change to the composition (a new baseline ver-

sion, a new overlay version, a new compose rule version) requires a `TxUpdateRuleSet` (designed; see Section 8) co signed by the role typed quorum, with a cool off window during which the regulator can repudiate. The "we accidentally made the rules looser" failure mode is structurally not available.

4.4 Invariant (iv): refusal code completeness and soundness

Statement. Every gate refusal carries a code from a stable, published taxonomy. Every named refusal in the taxonomy is reproducible from the on chain state by any third party with read access (*soundness*). Every admissibility failure produces some named refusal in the taxonomy; the gate does not fail silently and does not fail with an opaque or vendor specific code (*completeness*).

Mechanism. The taxonomy is part of the product surface, not an internal log convention. The full taxonomy is reproduced in Appendix A. Stability is by design: renames are a breaking change; deprecations carry an overlap window.

Consequences. A buyer's GC can write a control statement that references a refusal code by name. A regulator can recompute, from the on chain state, what the gate would have refused under any historical ruleset version, without operator cooperation. The proof is contractable, not narrative.

4.5 Invariant (v): consensus time non admit

Statement. A block that was refused admission at block height `H` cannot later be admitted at any block height `H' > H` against the same `ComposedRulesetHash` and the same canonical descriptor hash `H(d)`. The historical refusal cannot be retroactively rewritten to a permit. Conversely, an admission decided at block height `H` against `ComposedRulesetHash_H` is verifiable by any third party against `(H, ComposedRulesetHash_H, H(d))`, regardless of any later ruleset update or any later validator set change.

Mechanism. The ledger is hash linked: each block carries the cryptographic hash of its predecessor; the contents of any block are summarised by a tamper evident append only log whose root sits in the block header. A change to any historical record breaks the hash chain from that point forward, requiring remining every subsequent block and convincing every honest validator to accept the rewrite. Under the BFT honest supermajority assumption, this is infeasible. The deterministic policy decision engine and the deterministic descriptor canonicalisation mean a third party can independently rederive the same verdict from the recorded state, every time, on any machine.

Consequences. The audit record is checkable, not merely auditable. The proof is the on chain state and the deterministic evaluator, not the vendor's log. The trust given to the vendor is bounded by deterministic replay: even if every layer of the vendor's stack turned out to be flawed, a regulator who recomputes the verdicts catches a divergence within minutes of asking.

4.6 Invariant (vi): freshness binding (temporal admissibility)

Statement. Every PERMIT is bound to the freshness of the reference data it relied on (self exclusion register, consent register, sanctions list, age verification, affordability, hours of operation, do not call registry, telephone preference service). For each consulted register `r`, the proof carries `(RegisterID_r, VersionHash_r, AppliedAtUnix_r, CheckedAtUnix_r, AgeAtCheckSeconds_r, SLAWindowSeconds_r)`. The PERMIT is non admissible outside its named validity window. Re deciding the same action after a relevant register update (a new self exclusion against the recipient, a withdrawn consent, a tightened rule) yields REFUSE under the updated state, and the original PERMIT's window is effectively reduced to its `CheckedAtUnix_r` baseline for that register.

Mechanism. Three components together:

1. **Freshness named in the rule pack.** The deterministic policy reads the per register `AgeAtCheckSeconds_r` and refuses with `REFDATA_STALE` where the age exceeds the configured SLA. The SLA is part of the rule pack the regulator co signs.
2. **In window invalidation set.** Between pre certification and send, fresh sync events that flag a recipient add that recipient to the pre certified set's invalidation set. The per message step is "in the pre certified set, not in the invalidation set, token still fresh". A self exclusion registered ten seconds before send is honoured.
3. **Fail closed on staleness.** If the local replica's age exceeds the SLA, the gate refuses with `REFDATA_STALE`. The error direction is over suppress under partial failure. There is no "send anyway because the feed is down" path; that path does not exist by construction. A sustained upstream outage halts the operator's campaigns until the sync recovers.

The "without a real time lookup" question. A common, sceptical question from a regulated buyer's GC and from a regulator's technical reviewer is: how does the system verify that a subject is not self excluded without performing a real time lookup against the upstream register at the moment of every send. The answer is the freshness binding invariant. The per action proof carries a freshness stamp per register: a `VersionHash` identifying the snapshot of the register the gate relied on, an `AppliedAtUnix` timestamp identifying when that snapshot was applied to the local replica, a `CheckedAtUnix` timestamp identifying when the gate checked it, an `AgeAtCheckSeconds` value, and the contracted `SLAWindowSeconds`. The gate refuses any permit where `AgeAtCheckSeconds > SLAWindowSeconds`; the upstream register's signed `cursor_at` is also carried so an auditor can reconcile the operator's local replica with the regulator's published version log. Freshness is not the absence of a real time lookup; it is a named, contracted, stamped, fail closed substitute for one. The "are you sure" answer is "this is the verifiable record of how fresh the data was when we relied on it; here is the SLA; here is the on chain stamp; the operator could not have sent against a stale check because the gate refused".

Consequences. This closes the gap between compliant at decision and compliant at action: an action that was permitted at pre certification but whose reference data world has changed before send is refused at send. It closes stale authorisation replay.

It removes the operator's ability to set the freshness SLA wider than the regulator would accept, because the SLA is in the co signed rule pack. It makes the auditor's reconciliation a deterministic exercise against the upstream register's version log, not a request for operator side logs.

4.7 Decision time and holding over time, side by side

The six invariants partition into two columns:

Decision time (at the moment of admission)	Holding over time (across the history of the record)
(i) Non bypass	(iv) Refusal code completeness and soundness
(ii) Hash binding	(v) Consensus time non admit
(iii) Composed policy admissibility	(vi) Freshness binding (temporal admissibility)

Table 1. Six invariants in two columns. The decision time set says what the gate does at the block boundary; the holding over time set says what the record continues to mean as time passes.

The symmetry is deliberate. (i), (ii), (iii) say what the gate does at the moment of admission. (iv), (v), (vi) say what the record means after admission, as time passes, as registers update, and as auditors arrive. A compliance claim that holds only at one moment is fragile; the system's strength is that the record continues to support the same claim every time it is rechecked.

5. Threat model

The bounded claim defended by this section is the consensus rule property of Section 4 plus the dual authority composition plus the per action proof on the same ledger. Out of scope: the substance of the rules themselves (a regulator publishes an incorrect rule; a tenant's counsel writes a permissive overlay), network layer DDoS (a standard operational concern with standard mitigations), and social engineering of named signers outside the cryptographic gate (a governance problem; the engine reduces blast radius but does not replace governance).

We list the attacks an adversarial auditor or red team is expected to bring, and the structural defence the gate provides against each. The detailed adjudication is in Duke (2026d) and Duke (2026f); we summarise here.

5.1 Attribute and certification swapping

An attacker presents the gate with noncompliant content and tries to attach the verdict or attributes of a different, compliant piece (verdict swap; attribute injection; content swap after certification; cross creative re use). Defence: hash binding (invariant ii) plus per resolver signed attestations over `H(d)` (designed; the resolver's

swap point is the same as the signer's). Status: hash binding of the verdict, the per action proof, and the on chain violation record is exercised end to end in the reference embodiment (scenarios T6, T14, T15, T16). Per resolver signed attestations bound to `H(d)` are designed but not yet wired; the resolver runs in process within the trusted certifier service today.

5.2 Insider gaming

The operator's compliance leadership is incentivised on throughput or refusal rate targets. The AI evaluator's threshold is tuned to hit the target rather than the rule. A human signer signs without reading. A senior operator employee biases the engine. Defence: structural separation of duties (the generator is not the evaluator); the AI evaluator is one signer in a role typed quorum and cannot alone permit an action the deterministic policy floor refused; the deterministic policy floor catches anything the AI permits in error; human signers are biometrically and non repudiably bound to the action (`HUMAN_SIGNOFF_REQUIRED` consensus invariant routing; roadmap); the AI confidence threshold is itself part of the co signed rule pack and changing it requires the same dual authority co signature as any other policy change.

5.3 Black box or un owned generator

The operator buys a generative pipeline from a third party that will not expose model weights or prompts. The vendor's output lands directly in the operator's email or SMS gateway. The deciding factor is not whether the operator owns the generator; it is whether the operator holds an egress chokepoint through which the output must pass. The system enforces per piece on the output at that chokepoint and does not require generator side visibility. The trust dial modulates scrutiny per origin: less trusted generation raises the deterministic Layer 1 strictness, lowers the flag threshold for routing to the deep evaluator, and raises the statistical sample rate for deep review. Status: origin agnostic ingestion is built (the `NormalisedAction` wraps a `Descriptor` with `Origin in {human, service, ai, legacy, third party}` and `Mode in {observe, enforce}`); the trust dial implementation is a rule pack configuration, not an engine change.

5.4 Stale reference data

A recipient registers a self exclusion at 09:00; the campaign send fires at 09:05; the operator's pre certified audience was computed at 08:45. Defence: freshness binding (invariant vi) plus the in window invalidation set plus fail closed on staleness. Status: descriptor flags (`self_excluded`, `vulnerable`, `affordability_status`, `age_band`) are read by the deterministic policy baseline today (T2, T3 reproduce). The local replicated register subsystem and the `RefDataStamp` per consulted register are designed in Duke (2026c, Section 4) but not part of the Phase-3 reference binary; this is the next adoption critical add.

5.5 Collusion against the ledger

A sufficient subset of validators colludes to admit a noncompliant block or to rewrite history. Defence: Byzantine fault tolerance (`N = 3f + 1`; reference embodiment runs `N=4, f=1`). Multi party validator distribution across mutually distrusting parties (Kronaxis Limited holds a minority at every tier; client, auditor, and neutral notary hold the remainder). Status: the BFT property and the consensus time gate are built and reproduce on T5; multi party validator distribution is a deployment and governance step taken at first paying tenant onboarding, not an engine change.

5.6 Compromised host or resolver

A resolver host is compromised; the attacker substitutes its own resolver binary that reports `mandatory_elements_present = true` for every content descriptor. A signer host is compromised and the signing key is exfiltrated. The AI evaluator host is compromised and a different model is silently swapped in. Defence: separation of duties (compromising one role's host does not give the attacker the other role's key); TEE attested resolvers, signers, and evaluators (Intel TDX, AMD SEV SNP, AWS Nitro Enclaves; roadmap, the centrepiece of the trust hardening commission); reproducible builds (prerequisite for TEE attestation to be meaningful); HSM backed signer keys (Vault Transit for development; YubiHSM2 for early production; AWS CloudHSM or Azure Dedicated HSM for FIPS 140-3 Level 3 deployments; roadmap); consensus cross checks the signed result against the descriptor (the honest supermajority is the structural backstop).

5.7 AI evaluator nondeterminism abuse

The large language model evaluator's token output is non deterministic across runs. An attacker (or the operator under pressure) abuses this: "the model said approve last week and refuse this week on the same advert"; "the audit record is meaningless". Defence: the deterministic policy decision engine is the deterministic decision layer only; facts (including the AI's verdict) are computed outside the deterministic policy engine and fed in as attributes; the AI evaluator is one signer in the quorum and only ever adds a refusal that the deterministic policy engine permitted (never converts a deterministic refusal into a permit); the model's prompt, input, output, model identifier, model version, and inference parameters are recorded on chain (reproducibility moves from byte identity to evidence completeness for the fuzzy content portion); TEE bound model code and weights (roadmap). Status: the AI evaluator is wired as a quorum signer; the structured fact path stamps model fields today; TEE attestation of the inference stack is roadmap.

5.8 Policy or ruleset tampering

An attacker changes the active ruleset; the change permits actions the deposited regulator aligned ruleset refused. Defence: the composed ruleset is ledger resident; dual authority signature is enforced at the consensus gate; ruleset updates are co signed transactions with a cool off window; there is no in process override; the composer is conservative (a deny in either component is a deny in the composition; the overlay is

at least as strict as the baseline). Status: ledger resident registration, dual authority enforcement at consensus, per action proof binding the ruleset version, and app hash divergence detection across validators are built (T4, T10, T11). `TxUpdateRuleSet` with the draft, co sign, cool off, activate flow is designed but not wired; today ruleset updates rotate via a new versioned registration.

5.9 Residual structural risks

After every defence above is applied, the residual is bounded:

- The substance of the rules is not adjudicated by the engine. A regulator may publish an insufficient rule. A tenant's counsel may sign a permissive overlay. The engine enforces what is signed; it does not assess whether what is signed is the correct thing to sign.
- The AI evaluator is not bit replayable for the fuzzy content portion. The proof is evidence completeness against a TEE attested code and model hash, with the deterministic policy decision engine as the deterministic floor.
- Statistical assurance is statistical. The cheap classifier has a measurable false negative rate; the design publishes the sample rate, the confidence interval, and the aggregate per period.
- Sustained upstream outage halts campaigns by design. Fail closed on reference data staleness is the right error direction for compliance.
- Validator party distribution is a deployment decision and is honest about its current status (today all four reference validators run on Kronaxis development infrastructure).

6. Evaluation

The evidence base supporting the architecture and the security properties is the reference embodiment at HEAD. The evaluation reports four classes of evidence: scenario coverage, adversarial review, AI evaluator benchmark, and engineering hygiene.

6.1 Scenario coverage

The reference embodiment exercises 26 scenarios across the consensus, quorum, policy, and modes paths. Every scenario produces a pass or fail verdict against a stated invariant and is reproducible by re running `make test`. Representative scenarios include:

- **T2 (`SELF_EXCLUDED` refused).** A `sendPromo` against a `self_excluded` recipient is refused by the deterministic policy decision engine with the named code. Reproduces on every run.
- **T3 (`VULNERABLE` refused).** A `sendPromo` against a recipient with `vulnerable = true` is refused with the named code. Reproduces on every run.
- **T4 (overlay strict than baseline).** Where the per tenant overlay is stricter than the regulator baseline, the composition refuses an action the baseline alone would permit. Reproduces.

- **T5 (Byzantine bypass rejected).** A single Byzantine validator (within `f`) pre votes for an uncertified action; the honest three of four refuse; the block does not commit. Reproduces.
- **T6 (hash tamper rejected).** Mutating any byte of the canonical descriptor invalidates the signature; the gate refuses with `INVALID_SIGNATURE`. Reproduces.
- **T10 (per tenant overlay).** Two tenants with different overlays produce different verdicts for descriptors that differ only in `TenantID`. Reproduces.
- **T11 (cross tenant signer forgery rejected).** A signature from tenant Alpha's `RoleClient` signer on an attestation for tenant Beta is refused with `CROSS_TENANT_BINDING` or `INVALID_QUORUM`. Reproduces.
- **T12 (Observe records, never blocks).** In Observe mode, refusals are recorded as on chain violations; the action is not blocked at the egress. Reproduces.
- **T13 (Enforce returns HTTP 451 on refuse).** In Enforce mode, an HTTP egress refusal returns 451 with the `X-Compliance-Refusal-Code` header carrying the named code. Reproduces.
- **T14 (mode invariant verdict).** The same descriptor produces the same refusal code across Observe and Enforce modes. Reproduces.
- **T15 (origin invariant verdict).** The canonical descriptor hash is byte identical across `Origin in {human, service, ai, legacy, third party}`. Reproduces.
- **T16 (on chain violation cross validation).** The on chain violation record's descriptor hash matches the verdict's descriptor hash. Reproduces.

All 26 scenarios pass reproducibly on the four validator BFT reference cluster. The pass set is the binary acceptance gate for the consensus invariant and the structured policy layers.

6.2 Test hardening

A wider test hardening pass independent of the 26-scenario set covers unit tests, policy file validation, and rule pack scenario coverage:

- **105 of 105 unit test functions pass** (action, ingest, modes, registry, refusal, sdk, scenarios, and rulepack sanity tests).
- **13 of 13 policy files pass** via a new policy file validator. The pass uncovered and fixed two real `consumer_duty` rule pack bugs (five duplicate policy id markers in the baseline; a broken float literal in the overlay) that were not visible to the prior unit tests.
- **179 of 179 active rule pack scenario rows pass** via the `tools/scenario-runner` (8 rule packs, 6 justified pending rows documented in pack READMEs). The earlier 56-of-118 figure reported during initial scenario authoring was a real misalignment between policies and scenarios in the DRAFT stamped rule packs; the misalignment was closed by the gaps closing pass (`kc-gaps`, 2026-06-08) without weakening the rule packs.
- **21 of 21 policy source files validate** via the policy file static analyser (`tools/policy-validate`).
- **26 of 26 testnet scenarios pass** via the existing reference daemon binary on the BFT testnet.

The runner is a single `make test` (or `make test-fast` to skip the two minute BFT

testnet stage) and wires build, vet, unit, policy, scenario, and testnet stages with per stage pass or fail and a non zero exit on any failure.

6.3 Adversarial review (red and blue)

Three independent red teams (A: cryptographic and consensus surface; B: AI, content, and reference data; C: human, trusted execution, deployment, and insider) ran read only adversarial review of the Phase-3 reference implementation. A blue team adjudication produced a deduplicated, prioritised consolidated finding set. The headline numbers:

Severity	Count	Status
Critical	10	5 BUILT GAP (sprint scope fix); 5 ROADMAP GAP (named swap point)
High	12	11 BUILT GAP; 1 PARTIALLY CONFIRMED (lights up when AI signer wired)
Medium	13	8 BUILT GAP; 5 PARTIALLY CONFIRMED (light up when corresponding subsystem ships)
Low	8	5 ALREADY DEFENDED; 3 BUILT GAP (ops nuisance)

Table 2. Red and blue team adjudicated findings by severity and disposition (n=43, deduplicated across three independent reviews).

The red teams found one set of CRITICAL findings the previous threat model had not named (the `TxSubmitAction` ActionID decoupling, a single line cross check fix; the `TxRecordViolation` ingester signature gap, honestly acknowledged in the threat model's roadmap section but elevated to CRITICAL by the red teams because the headline integrity claim depends on the binding existing). The remaining ROADMAP GAP rows align with the named trust hardening commission. The blue team's prioritisation produces a MUST FIX BEFORE PILOT list of seventeen items (all single sprint scope) and a MUST FIX BEFORE PRODUCTION list of twelve items aligned with the existing roadmap.

The exercise validates the threat model where it was right and tightens it where it over claimed. The full adjudication is at Duke (2026f).

6.4 AI evaluator benchmark (n=700)

The augmenting AI evaluator (a locally hosted, quantised, mid sized open weights large language model on the operator's own GPU; zero remote inference) was bench-

marked against a stratified dataset of 700 records labelled by the deterministic policy oracle. The headline numbers (refined prompt run; structured fact scenarios):

Metric	Value
Overall (joint label) accuracy	0.9886
Binary comply versus refuse accuracy	0.9886
Refuse precision	0.9916
Refuse recall	0.9916
Refuse F1	0.9916
Parse and schema errors	0 of 700
Mean wall time per evaluation	~3.12 seconds
Mean completion tokens	~40

Table 3. AI evaluator benchmark on a 700 record stratified dataset, refined prompt run, structured fact scenarios.

The evaluator is fit as one signer in the role typed quorum and is **not** fit as the sole gate. The two failure modes in the refined prompt run are: conservative bias on responsible gambling and transactional kinds (four cases; safe failure mode; refuses an action the policy would permit but never permits one the policy would block), and an `hour_local == 21` boundary condition where the model reads the natural language operating window "09:00 to 21:00" inclusively and permits at 21:00 (four cases; the only permissive failure mode in the run; in dual authority use the deterministic policy signer catches this with `OUTSIDE_OPERATING_HOURS`). The result validates the dual authority design at a measurable level: a fast LLM evaluator is reliable enough to be a useful rederivation signer (98.86%) but not reliable enough to be the sole gate (one out of every ~58 refuse cases would slip through). The deterministic policy signer is the floor; the AI is the independent verifier; the quorum's AND composition is what makes the property non trivial.

Hardware utilisation evidence: the dataset generation phase drove GPU 1 at 83.2% mean utilisation; the benchmark phase drove GPU 1 at 96.7% mean utilisation with 100% median during inference batching. Maximum resident memory was 22,641 MiB. GPU 0 was idle throughout.

6.5 Engineering hygiene

The reference embodiment ships with:

- **Canonical encoding by length prefixed binary**, not JSON, for descriptor, envelope, attestation, and proof artefacts. The format is deterministic across language versions and platforms.
- **Stable refusal code taxonomy** (Appendix A); renames are a breaking change by design.
- **App hash divergence detection** at the finalisation hook: a validator that diverges in the ruleset registry or in the policy evaluation produces a mismatching app hash and is evicted at the next height.

- **Origin agnostic ingestion** at the `NormalisedAction` boundary (`Origin in {human, service, ai, legacy, third party}`); descriptor hash byte identity verified by T15.
- **Mode invariant verdict** across Observe and Enforce (T14).

The engineering surface is small (the Phase-3 reference binary is on the order of 4,000 lines of memory safe systems language source), deliberately narrow, and structured around the swap points named in the trust hardening commission.

7. Deployment

The same engine, the same ruleset, and the same proof support two architecturally distinct deployment modes (Observe and Enforce), a retrofit pattern for an existing pipeline, and a transparency mirror story for public verifiability.

7.1 Observe mode (out of band)

Observe mode ingests actions that have already happened, or that are about to happen, from sources the firm does not want to put a synchronous gate in front of. The engine certifies, proves, and surfaces refusals as alerts. It does not, and cannot, block. Typical adapters: application log taps; database change data capture; message bus mirrors (Kafka, NATS, Pub/Sub); egress mirrors from web proxies, email gateways, and SMS aggregators; webhook receivers; human activity logs.

Observe mode does not need engineering integration into the action producing system. It needs only a tap. It is sellable into production on the day a buyer says yes because it cannot affect the production path. The output for the buyer is provable continuous audit, real time alerts for noncompliant actions, and a regulator faceable record that does not depend on the operator of any source system. This is the adoption wedge.

The reference embodiment ships Observe mode with a four line retrofit at startup, engine off the critical path, the `MemoryAlertSink` for refusals, and a signed `TxRecordViolation` on chain for every refusal. Scenario T12 reproduces the property: refusals are recorded; the underlying action is delivered (because Observe cannot block); the on chain record carries the descriptor hash and the refusal code.

7.2 Enforce mode (inline)

Enforce mode sits on the action's critical path and physically prevents the action when the gate refuses. Same engine, same ruleset, same proof, but the action does not execute on a refusal. Typical enforcement points: API gateway plug ins (Envoy, Kong, AWS API Gateway, NGINX) for outbound calls; sidecar interceptors for service to service traffic in a mesh; queue interceptors for asynchronous work; database pre commit triggers; batch pre processors for bulk campaigns; SDK call sites for actions originated inside the firm's own code.

The reference embodiment ships Enforce mode with a two line retrofit at startup, engine on the critical path, HTTP 200 on clean send, HTTP 451 with `X-Compliance-`

`Refusal-Code` header on refuse, and the JSON `Verdict` body with `proof_ref{index, outcome: REFUSED, log_root_hex}` in the response. Scenario T13 reproduces the property: a refused send returns 451; no backend hit occurs on refuse; verified by `len(server.Delivered()) == 1` in the worked retrofit example (Duke, 2026h).

The retrofit example (Duke, 2026h) demonstrates the operative difference between the three deployment shapes (no gate; Observe; Enforce) on a realistic gambling CRM `/send` endpoint:

Shape	Self excluded send	Engine	Critical path
No gate	HTTP 200 (regulatory breach baseline)	not wired	not applicable
Observe	HTTP 200 (delivered) plus `TxRecordViolation` on chain with `SELF_EXCLUDED` and `did:kxc:regulator-baseline:kronaxis` as `refused_by`	wired	off
Enforce	HTTP 451, `X-Compliance-Refusal-Code: SELF_EXCLUDED`, `Verdict` body with `proof_ref{outcome: REFUSED, log_root_hex}`	wired	on

Table 4. Three deployment shapes on a realistic gambling CRM `/send` endpoint. The CRM `SendHandler` is byte identical across all three shapes; only the wiring at startup changes.

The CRM `SendHandler` is byte identical across all three shapes; only the wiring at startup changes. Zero core module edits are required to retrofit an existing pipeline.

7.3 Public verifiability (external transparency log mirror)

Public verifiability without requiring an auditor to operate a validator is provided by mirroring every committed quorum attestation row and every per action proof to an external, publicly readable transparency log. The external log’s inclusion proof is itself hashed and committed back to the consensus ledger at the next block, so the consensus ledger remains the canonical anchor and the external log remains the consumer readable mirror. The reference embodiment ships the local tamper evident append only log internal to the cluster; the external mirror is roadmap (Duke, 2026b, Section 7) and the swap point is named at the inclusion proof boundary.

7.4 Validator topology

The reference embodiment runs four validators on Kronaxis Limited development infrastructure. The production target topology is:

- **Pilot tier (`N=4, f=1`).** One Kronaxis Limited operated validator. One client operated validator. One auditor operated validator. One neutral notary operated validator. Kronaxis Limited holds a minority at every tier; forging a block requires Kronaxis plus two of the other three to collude across commercial, regulatory, and audit boundaries.
- **Mature tier (`N=7, f=2`).** One Kronaxis Limited validator. Two client cooperative validators (the paying tenant plus a second). One auditor. One neutral notary. Two regulator aligned read only validators.

Validator set composition is a deployment time decision and is auditable on chain. Validator additions and evictions are `2f+1` supermajority governance acts, recorded as transactions. A buyer or a regulator can read the genesis configuration and the validator change history from the ledger and confirm who controlled the cluster at any historical block height. Validator keys rotate annually; baseline signer keys rotate six monthly; recovery uses a 2-of-3 DID quorum so a single party cannot unilaterally rotate to itself.

8. Limitations and honest residual

The discipline of this paper is that every claim is held to what the built reference embodiment delivers, and every roadmap row is named for what it is. This section consolidates the limitations.

8.1 What the bounded claim does not assert

- That the deposited regulator baseline is the legally correct interpretation of the regulator's rules. The substance of the baseline is the regulator's content (or content the tenant has chosen to align with the regulator's published material). The engine enforces it; the engine does not adjudicate it.
- That the deposited tenant overlay is the legally adequate set of additional controls for the tenant's particular risk position. The substance of the overlay is the tenant's counsel's content.
- That no regulator action will ever later find that the deposited ruleset was incomplete, or that some action the deposited ruleset permitted should, in retrospect, have been refused. The deposited ruleset is the firm's and its advisers' responsibility.
- That the Kronaxis side software is bug free. The cryptographic invariants stand; software defects in the policy authoring tools, the audit console, or the adapters can still occur and are managed by the same software quality processes any vendor uses.

8.2 The AI evaluator caveat

The augmenting AI evaluator is not bit replayable in the same way the deterministic policy engine is. A regulator cannot, in general, rerun the LLM on their own hardware and be sure of getting the exact same output token for token (model stack nondeterminism includes batching, sampling, library version, hardware ordering). The mitigations are layered:

1. The AI evaluator is one signer in the quorum, not the sole authority. The deterministic policy verdict is required separately. The AI only ever adds a refusal that the deterministic policy engine permitted; it cannot, architecturally, permit an action that the deterministic policy engine refused.
2. The AI's prompt, input, output, model identifier, and model version are all recorded on chain as part of the per action proof. Reproducibility moves from byte identity to evidence completeness for the fuzzy content portion.
3. The model is bound, in the production target, to a TEE attested code and weights hash. A silently swapped model produces a model identifier the auditor can see and a measured hash the attestation does not match.
4. The deterministic policy engine is the deterministic floor. The fuzzy content portion is narrowed continuously: a class of mistakes the AI makes repeatedly becomes a deterministic rule the moment a human notices it.

The honest claim for the LLM evaluable portion is narrower than for the deterministic policy path: the proof is the on chain record of what the model saw and said, against an attested code and model hash, with the deterministic policy engine as the deterministic floor. That is not nothing; it is also not the same as the bit identical guarantee the deterministic policy engine gives.

8.3 The trust hardening commission (what is built today; what is on the build queue)

Component	Status today
Unforgeable digital signature primitives across certifiers and validators	Built; software held keys today (see HSM row below)
Canonical action descriptor and hash binding	Built; origin invariant (T15) and mode invariant (T14) byte identity proven across runs
Hash linked tamper evident ledger	Built; four validator deterministic BFT reference cluster
Tamper evident inclusion proofs (per action)	Built; local tamper evident append only log plus round trip verified inclusion proofs
BFT consensus gate on uncertified actions	Built; T5 reproduces
Deterministic policy evaluator and replay	Built; deterministic by construction; reproduced across validators on every commit
Dual authority composed ruleset	Built; T4 and T10 reproduce

Component	Status today
Tenant required role typed quorum	Built; T11 cross tenant binding rejected
Refusal code taxonomy (named, contractable)	Built; stable across versions; Appendix A
Origin agnostic ingestion, Observe and Enforce modes	Built; T12, T13, T14, T15 reproduce
AI evaluator on locally hosted LLM, augmenting the deterministic policy engine on fuzzy content	Built with documented caveat (Section 8.2); 98.86% / 99.16% on n=700
HSM backed signer keys	Roadmap (commissioned); swap point `quorum.SignerKey.Sign`
TEE attestation of the signer	Roadmap (commissioned); target enclaves: Intel TDX, AMD SEV SNP, AWS Nitro Enclaves
Client hosted signer (Layer 1, strongest answer to the trust the box question)	Roadmap (commissioned); signer interface is the swap point
Reproducible builds for the signer	Roadmap (commissioned); prerequisite for TEE attestation to be meaningful
External transparency mirror (publicly readable append only log)	Roadmap; swap point at the inclusion proof boundary
Persistent ledger state on disk	Roadmap; production hardening pass
`N-of-M` governance signatures on the ledger configuration	Roadmap; not production acceptable for a paying tenant until shipped
Audit console (per tenant subdomain, GC facing drill down, signed proof PDF)	Roadmap; underlying audit substrate (the ledger) is shipped
Reference data subsystem (local replicated registers, `RefDataStamp`, in window invalidation set)	Roadmap; next adoption critical add
Biometric mobile signer for `HUMAN_SIGNOFF_REQUIRED`	Roadmap

Table 5. Trust hardening commission. Built rows hold today; roadmap rows are named with a stated swap point.

The structural property holds today at the consensus rule and cross tenant binding layers and at the dual authority composition layer. Full operational removal of platform operator capability requires the client hosted Layer 1 signer, the TEE attested signing binary, and the multi party validator distribution. Each is a named row with a swap point.

8.4 What is out of scope

- Rule authorship attacks (a captured regulator; a bribed tenant counsel; an incompetent rule pack maintainer) are governance problems. The engine reduces the blast radius but does not replace governance.
- Network layer DDoS, mempool flooding without a fee market, and other standard operational concerns are noted as roadmap and not detailed here.

- The internal cryptographic primitives of the consensus engine are upstream and treated as trusted.

9. Conclusion

A regulated organisation that deploys autonomous AI agents and automated pipelines must be able to prove, after the fact, that each regulated action conformed to the applicable rules, to a third party, without the operator's cooperation. Existing answers reduce to operator trust, single authority attestation, or post hoc verification of an opaque record. None gives the buyer's General Counsel or the regulator a position from which they can check, rather than trust, the operator.

The method consolidated and formalised in this paper combines five elements that, taken together, give the buyer that position. The action descriptor is canonicalised and hash bound. The operating ruleset is the deterministic composition of a regulator aligned baseline and a per tenant overlay, both authored under different authority and both ledger resident. The activation quorum is role typed and architecturally requires a tenant controlled signer; cross tenant binding is enforced at the consensus layer. The certification check is a rule of the Byzantine fault tolerant consensus engine; a single Byzantine validator within the configured fault threshold cannot cause an uncertified action to commit. The per action proof co resides with the policy and the quorum on the same ledger, and every PERMIT is bound to the freshness of the reference data it relied on, with `REFDATA_STALE` as the fail closed code outside the contracted SLA.

The six gate invariants (three decision time, three holding over time) make the assurance state explicit. The reference embodiment exercises 26 scenarios that reproduce the structural properties. Three independent red teams and a blue team adjudication produced a prioritised finding set against the engineering surface; the load bearing critical issues are sprint scope fixes or named trust hardening commissions. The AI evaluator benchmark places the augmenting model at 98.86% joint label accuracy and validates the dual authority design as the right architectural choice: the AI is reliable enough to be a useful rederivation signer but not the sole gate.

The freshness binding invariant closes the gap between compliant at decision and compliant at action and provides the formal answer to the practical question of how a self exclusion or a withdrawn consent is honoured without a synchronous lookup at the moment of every send. The answer is named freshness, stamped on the proof, contracted in the rule pack, fail closed under partial failure, and reconcilable against the upstream register's published version log.

The system is built, the consensus property is built, the dual authority composition is built, and the reference embodiment exercises the invariants reproducibly. The trust hardening commission (HSM backed keys; TEE attestation of the signer; client hosted Layer 1 signer; reproducible builds; multi party validator distribution; the external transparency log mirror; persistent state; `N-of-M` governance; the reference data subsystem; the biometric mobile signer) is named, commissioned, and on the build queue. Every external statement is held to which layer holds today.

The honest one line positioning is the same as it has been since the project's inception. **Check us, do not trust us.**

10. Acknowledgements

This work draws on the published rule sets, codes of practice, and statutory guidance of the United Kingdom Gambling Commission, the Financial Conduct Authority, the Information Commissioner's Office, the General Pharmaceutical Council, the Office of Gas and Electricity Markets, the Solicitors Regulation Authority, the Advertising Standards Authority, and the Committee of Advertising Practice. The author acknowledges those sources without implying any endorsement.

The author acknowledges the independent red and blue review participants (cryptographic and consensus surface; AI, content, and reference data; human, trusted execution, deployment, and insider) whose adversarial work shaped the threat model and the trust hardening commission of Section 8.3. The participants are named in Duke (2026f).

The author acknowledges the open standards and open source communities behind the deterministic authorisation policy language family, the Byzantine fault tolerant consensus engine family, and the public transparency log family on whose primitives the method relies.

11. References

Attested Intelligence. (2025). *Attested Governance Artifacts.* USPTO patent pending Serial 19/433,835. https://attestedintelligence.com/

Bagheri, A., *et al.* (2025). A Blockchain Monitored Agentic AI Architecture for Trusted Perception Reasoning Action Pipelines. *arXiv preprint* 2512.20985. https://arxiv.org/abs/2512.20985

Cerbos Project. (2024). *Cerbos: Decoupled Authorisation for Cloud Native Applications.* https://www.cerbos.dev/

Credo AI. (2024). *AI Governance Platform.* https://www.credo.ai/

Duke, J. (2026a). *Provably Compliant Autonomous Agents: A Defensive Publication.* Kronaxis Limited. 7 June 2026. (Defensive publication, final.)

Duke, J. (2026b). *Kronaxis Compliance: Trust and Assurance.* Kronaxis Limited. 7 June 2026.

Duke, J. (2026c). *Verified Compliance: Full System Specification.* Kronaxis Limited. 6 June 2026.

Duke, J. (2026d). *Kronaxis Compliance: Threat Model and Attack Resistance.* Kronaxis Limited. 7 June 2026.

Duke, J. (2026e). *Verified Compliance: Prior Art Sweep.* Kronaxis Limited. 6 June 2026.

Duke, J. (2026f). *Kronaxis Compliance: Red and Blue Adjudication Report.* Kronaxis Limited. 7 June 2026.

Duke, J. (2026g). *Kronaxis Compliance: AI Evaluator Benchmark.* Kronaxis Limited. 7 June 2026.

Duke, J. (2026h). *Kronaxis Compliance: Worked Retrofit Example (Gambling CRM).* Kronaxis Limited. 7 June 2026.

Financial Conduct Authority. (2024). *Consumer Credit Sourcebook (CONC).* United Kingdom. https://www.handbook.fca.org.uk/handbook/CONC.pdf

Holistic AI. (2024). *AI Governance Software Platform.* https://www.holisticai.com/

Hyperledger Project. (2024). *Hyperledger Fabric: A Permissioned Distributed Ledger.* The Linux Foundation. https://www.hyperledger.org/projects/fabric

IBM. (2024). *watsonx.governance: AI Lifecycle Governance.* IBM Corporation. https://www.ibm.com/products/watsonx-governance

Information Commissioner's Office. (2024). *Direct Marketing Guidance.* United Kingdom. https://ico.org.uk/

Laurie, B. (2013). *Certificate Transparency.* RFC 6962. Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc6962

Laurie, B., Messeri, E., & Stradling, R. (2021). *Certificate Transparency Version 2.0.* RFC 9162. Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc9162

Microsoft. (2026). *Agent Governance Toolkit: Open Source Runtime Security for AI Agents.* MIT licence, released 2 April 2026. https://opensource.microsoft.com/blog/2026/04/02/

OneTrust. (2026). *AI Governance Platform.* https://www.onetrust.com/

Open Policy Agent Project. (2024). *Open Policy Agent: Policy Based Control for Cloud Native Environments.* Cloud Native Computing Foundation, Apache-2.0. https://www.openpolicyagent.org/

OpenPort Project. (2026). *OpenPort Protocol: A Security Governance Specification for AI Agent Tool Access. arXiv preprint* 2602.20196. https://arxiv.org/abs/2602.20196

Oso Project. (2024). *Oso: Authorisation as a Service.* https://www.osohq.com/

ServiceNow. (2026). *Governance, Risk and Compliance plus AI Control Tower.* https://www.servicenow.com/products/governance-risk-and-compliance.html

South, T., *et al.* (2026). Before the Tool Call: Deterministic Pre Action Authorisation for Autonomous AI Agents. *arXiv preprint* 2603.20953. (Open Agent Passport / OAP specification.) https://arxiv.org/abs/2603.20953

United Kingdom Gambling Commission. (2024). *Licence Conditions and Codes of Practice.* https://www.gamblingcommission.gov.uk/licensees-and-businesses/lccp

Appendix A. Refusal code taxonomy

The full stable taxonomy of refusal codes returned by the gate. Every code is reproducible from on chain state by any third party with read access (soundness); every admissibility failure produces some code in the taxonomy (completeness). Renames are a breaking change by design.

Code	Stage	Meaning
`UNCERTIFIED_BYTECODE`	Consensus	The action carries no quorum attestation referencing `H(d)` against an active composed ruleset.
`ATTESTATION_REVOKED`	Consensus	A signer's attestation has been revoked since admission was sought.
`ATTESTATION_EXPIRED`	Consensus	The attestation's validity window has closed at the block height under consideration.
`SCOPE_MISMATCH`	Consensus	The action class is outside the scope of the attestation.
`UNKNOWN_CERTIFIER`	Consensus	A signer on the attestation is not registered in the certifier registry.
`CERTIFIER_REVOKED`	Consensus	A signer on the attestation has been revoked from the certifier registry.
`INVALID_SIGNATURE`	Consensus	A signature on the attestation does not verify against the canonical descriptor.
`INVALID_QUORUM`	Consensus	The signature set does not satisfy the active quorum policy.
`ROLE_MISMATCH`	Consensus	A required role type is missing from the signature set.
`CROSS_TENANT_BINDING`	Consensus	A `RoleClient` signer's tenant identifier does not match the descriptor's tenant identifier.
`SELF_EXCLUDED`	Deterministic policy	The recipient is on the self exclusion register for the vertical.

Code	Stage	Meaning
`VULNERABLE`	Deterministic policy	The recipient is flagged vulnerable; the action kind is restricted under the rule pack.
`OUTSIDE_OPERATING_HOURS`	Deterministic policy	The action falls outside the contracted operating hours window.
`AFFORDABILITY_BLOCKED`	Deterministic policy	Affordability data prohibits the action against this recipient under the rule pack.
`AGE_UNVERIFIED`	Deterministic policy	The recipient’s age has not been verified to the standard required by the rule pack.
`AD_CODE_VIOLATION`	Content	The creative violates the applicable advertising code (CAP, BCAP, or vertical specific).
`CLIENT_OVERLAY_STRICT`	Composition	The per tenant overlay refuses an action the baseline alone would permit.
`MANDATORY_ELEMENT_MISSING`	Content	The creative omits an element the rule pack marks mandatory (a required disclosure, a required hyperlink, a required age warning).
`PROHIBITED_ELEMENT_PRESENT`	Content	The creative contains an element the rule pack marks prohibited.
`BANNED_PHRASE`	Content	The creative contains a phrase the rule pack lists as banned.
`WATERSHED_VIOLATION`	Content	The creative or its delivery breaches a watershed restriction.
`TARGETING_MISMATCH`	Targeting	The action targets a segment outside the rule pack’s allowed targeting envelope.

Code	Stage	Meaning
`REFDATA_STALE`	Freshness	The local replica of a consulted register exceeds the contracted `SLAWindowSeconds` SLA at the moment of admission.
`HUMAN_SIGNOFF_REQUIRED`	Routing	The rule pack requires a human signer for this action; no signer has signed.
`KIND_CONTENT_MISMATCH`	Composition	The declared action kind and the canonical payload class do not match.

Table 6. Stable published refusal code taxonomy. The taxonomy is part of the product surface, not an internal log convention.

Appendix B. Source repository layout

The functional layout of the source repository, expressed without disclosing the implementation language identifiers, internal module names, or wire protocol surface that remain trade secret:

```
kronaxis-compliance
+-- consensus/      # BFT consensus engine integration and pre execution hook
+-- policy/           # Deterministic policy evaluator and rule pack loader
+-- quorum/       # Role typed signer interface, certifier registry, attestation store
+-- proof/            # Per action proof writer, inclusion proof verifier
+-- refdata/       # Reference data replica subsystem (registers, freshness stamps)
+-- modes/            # Observe and Enforce mode adapters
+-- ingest/       # Origin agnostic action ingestion (NormalisedAction boundary)
+-- refusal/          # Stable refusal code taxonomy
+-- canonical/        # Length prefixed binary canonicalisation
+-- tools/
|   +-- scenario-runner/   # Rule pack scenario harness
|   +-- policy-validate/   # Policy file static analyser
|   +-- kc-export/         # Per action evidence export
|   +-- kc-verify/         # Third party inclusion proof verifier
+-- examples/
|   +-- retrofit/          # Worked retrofit example (gambling CRM)
+-- tests/            # Scenario tests, unit tests, BFT testnet runner
+-- docs/         # Public facing documentation set (the references of Section 11)
```

The on the wire protocols, the exact canonicalisation rule, the per vertical rule pack content, and the deployment topology are not reproduced in this paper; the layout above is sufficient for a reviewer to follow the architecture of Section 3 and to locate the corresponding tests of Section 6.

Provably Compliant Autonomous Actions: a consensus enforced, dual authority certification gate with freshness bound per action proofs. White paper version 1.0, 7 June 2026. British English. Author: Jason Duke, Kronaxis Limited, United Kingdom. Correspondence: jason@kronaxis.co.uk. ORCID placeholder pending. Paper text under Creative Commons Attribution 4.0 International. Implementation specifics withheld as Kronaxis trade secret. Operator deposits to Zenodo (DOI placeholder pending).