Nick Clark · Published April 25, 2026 · PDF

How to Distribute Signed AI Policy That Agents Enforce Offline

You have autonomous agents running across cloud, edge, and intermittently connected environments, and you need a policy that they cannot quietly disable, downgrade, or outrun when the network is gone. This guide describes an architecture for distributing cryptographically signed, externally governed policy objects that each agent verifies and enforces locally, before it acts, even offline. The approach is disclosed in United States Patent Application 19/561,229 (it is a described architecture, not a shipping library), and its home inventive step is the Cryptographic Governance inventive step.

What You Are Building

You are building a way to ship a governance policy to autonomous agents so that each agent enforces it itself, at the moment it is about to act, without phoning home to a central controller. The policy is signed and immutable. The agent treats it as a hard precondition: no execution context is instantiated unless the applicable policy has been resolved, verified, and found to authorize the specific action. When the agent is offline, the enforcement still holds, because the authority travels with the agent as verifiable data rather than as a live service call.

This solves a concrete problem. If you run agents that can execute code, mutate their own state, delegate tasks, or spawn copies of themselves, then the interesting failure mode is not "the agent made a bad decision." It is "the agent, or something acting through it, removed the rule and kept running." You want a design where removing or weakening the rule is itself a blocked action, and where a stale copy of the rule cannot be replayed after you have revoked it.

The disclosed architecture, described in United States Patent Application 19/561,229, treats governance as a deterministic cryptographic precondition to execution, mutation, delegation, and propagation. This guide walks through how that architecture is put together and how you would approach building it yourself. It does not ship you an implementation.

Why the Obvious Approaches Fall Short

The common ways to govern agent behavior each leave a structural gap when the agent is autonomous, distributed, or offline.

Embedding the rules in the agent's code or prompt. This is the default, and it is the weakest against self-modification. When the constraint lives inside the artifact that can rewrite itself, an agent (or an adversary operating through it) can alter, disable, or reinterpret the constraint through an update, a fork, or a serialization round-trip. The rule and the thing the rule governs share a mutation surface.

Enforcing through a central controller or gateway. Routing every action through a trusted runtime works while the agent is online and inside that runtime. It degrades the moment the agent migrates to another substrate, runs at the edge, or loses connectivity. Centralized enforcement also concentrates a single point of failure and expands attack surface, and it does not travel with a mobile agent.

Relying on alignment scoring, intent modeling, or outcome prediction. These infer whether an action is probably acceptable from the model's internal state. They are probabilistic, hard to audit, and sensitive to interpretability limits. They do not give a deterministic guarantee that a prohibited action class cannot be instantiated.

Auditing after the fact. Logs and monitoring detect violations after execution has occurred. For an autonomous agent that can act quickly and cause irreversible effects, post-hoc detection is not prevention.

There is also a distribution-specific gap. Even signed policies can be defeated by replay and downgrade: an agent holds onto an old, still-signed policy after you have issued a stricter successor, or an adversary re-presents a revoked version. Any offline design has to answer "how does the agent reject authority that is authentic but stale?" without a network round-trip.

The Architecture

The disclosed approach externalizes the policy from the agent and makes verification of that external policy a gate that execution must pass through. Every mechanism below traces to the filing.

The policy object is a standalone, signed, immutable authority. Policy is not configuration or advisory guidance embedded in the agent. It is a separate, machine-readable governance object that is independently storable, transmissible, resolvable, cacheable, and verifiable without any particular agent. In the disclosed structure it carries: a canonical alias binding (a stable identifier), a policy body defining permitted and prohibited action classes, a verification field holding authentication material bound to at least the alias binding and the policy body, a scope declaration (which agent classes, action classes, substrate classes, and trust zones it applies to), a validity and

freshness component, and an enforcement class field. Authenticated content is immutable by default; you do not edit a policy in place. Governance changes by issuing a successor object, not by mutation.

Agents reference policy by canonical alias, never by embedding it. The agent object carries a policy reference field containing one or more canonical aliases. The alias is a pure reference: it confers no authority by its presence. At runtime the alias is dereferenced to obtain the actual policy content plus provenance sufficient to verify it. This indirection is the pivot that makes offline distribution and later supersession possible, because you can publish a new authoritative object under the same alias without touching any agent.

A governance gate sits in front of execution. When the agent proposes an action, the proposal goes to a governance gate: a deterministic checkpoint that conditions instantiation of an execution context on verification of the applicable policy and an authorization determination under it. The gate can live inside the substrate, as middleware, as a verifier library, or as a validation service. The point is that no execution instance exists before the gate issues a permit. Denial is a valid, first-class outcome, not an error to be retried around.

Resolution, then freshness filtering, then verification, then authorization. The disclosed runtime pipeline is ordered. The gate extracts the required aliases and issues a resolution request to a resolver (the filing calls it a Dynamic Alias System, and notes it may be a scoped registry, adaptive index, or distributed naming substrate). Resolution returns a candidate set. Before cryptographic verification, the candidate set is filtered by freshness constraints: a validity-window filter (notBefore / notAfter semantics against an evaluation time basis), a revocation check against resolved revocation artifacts, and an anti-rollback evaluation. Only the surviving candidates are cryptographically verified for authenticity and integrity. Then the gate evaluates applicability (scope, and a re-check of freshness at authorization time) and finally authorizes the specific action class against the policy body. If every required alias

resolves, survives filtering, verifies, and authorizes, the gate emits a permit; otherwise it denies and no execution context is instantiated. Doing freshness filtering before verification means the system can deterministically reject stale or revoked candidates without spending verification on ineligible ones.

Freshness, revocation, and anti-rollback are what make offline enforcement safe. This is the part that answers the replay-and-downgrade problem. Validity windows let a policy be treated as non-authoritative before activation and after expiration regardless of authenticity. Revocation acts as negative authority: a revocation artifact renders a policy non-authoritative even if it is authentic and in-window. Anti-rollback prevents an agent from relying on an older instance once a newer authorized successor exists. The filing describes several expressions of anti-rollback: monotonic version indicators tied to the alias, required verifiable continuity references linking a successor to its predecessor (a candidate lacking the required continuity reference is non-authoritative even if authentic), a policy-defined minimum acceptable version, and a latest-known-good fingerprint stored in the agent's embedded memory or in an audit record that acts as a checkpoint against which later candidates are judged. Because that checkpoint travels inside the agent, downgrade rejection works with no network.

Offline operation uses cached policy under revalidation rules. In intermittently connected environments the pipeline may rely on cached policy objects and previously verified authority fingerprints. Cached authority is accepted only if authenticity, scope, validity, freshness, and anti-rollback requirements still hold under the revalidation policy. If required authority cannot be verified because it expired, was revoked, went stale, or cannot be revalidated, instantiation is denied rather than executed optimistically. Offline never means "assume yes."

Changing the rule requires quorum, and the change is itself verified. You do not want a single party silently weakening governance. The disclosed override mechanism requires a plurality of authorized participants to co-sign a replacement

policy object satisfying a quorum rule (the filing describes a threshold of at least two distinct participants in embodiments). The override object carries the co-signatures and a continuity reference linking it to the object it supersedes. It is published through the alias system under the existing alias, and prior instances are marked superseded or revoked. At runtime the gate does not trust an override because it appeared under the alias; it verifies authenticity, confirms the quorum is satisfied by the co-signatures, and validates the continuity reference back to the prior authoritative instance. If any check fails, the override is non-authoritative and the prior policy keeps controlling.

Distribution is decentralized and asynchronous. Because agents reference authority by alias, updates propagate by publishing new authoritative instances rather than mutating agents. The filing describes federated registries, content-addressable stores, distributed ledgers, replication, and gossip dissemination as options, with no single node required as global authority. Each node independently applies the same verification rules. Dissemination can be scoped, so the same alias can resolve to different authoritative instances per trust domain or substrate class for staged rollout.

An append-only, integrity-chained audit ledger records what happened. Every governance-relevant event (resolution outcomes, verification results, freshness and revocation and anti-rollback determinations, permits, denials, override approvals, trust degradation, quarantine) is written to an append-only ledger whose entries are cryptographically linked into an integrity chain so that removal, modification, or reordering is detectable. Per the filing, the audit ledger does not grant authority and runtime gating does not depend on successful logging; it is tamper-evident evidence, and it treats non-execution outcomes as first-class results.

Optional reinforcements. The filing also describes fallback enforcement agents that watch governance events across substrates as defense-in-depth (they do not sit in the critical authorization path and do not replace gating), and a keyless embodiment that establishes authority through verifiable continuity of governance evidence and trust-slope validation where persistent private keys are unavailable.

How to Approach the Build

Follow the grain of the pipeline. These steps mirror the disclosed structure; you are implementing them, not installing them.

1. **Define the policy object schema first.** Before any enforcement code, fix the wire format for a policy object: alias binding, policy body (an explicit machine-interpretable list of permitted and prohibited action classes), verification material, scope declaration, a validity and freshness component with notBefore / notAfter, and an enforcement class. Make it independently serializable and verifiable. This schema is the contract the rest of the system depends on.
2. **Make policy content immutable and addressed by content.** Decide how immutability is enforced (content-addressed storage, hash binding, signature binding, or a combination). Establish that a change means a new object with a new fingerprint, never an in-place edit. This is what later lets anti-rollback and audit work.
3. **Put an alias field in the agent, and nothing authoritative.** The agent object holds canonical aliases in its policy reference field. Resist the temptation to cache the policy body as authoritative inside the agent. A cached copy is allowed as an optimization for offline use, but it is only ever accepted after passing the same verification and freshness checks.
4. **Build the resolver.** Implement alias resolution that returns candidate policy objects with provenance. Start with a single scoped registry; the architecture generalizes to federated or gossip-based dissemination later. Have the resolver accept an evaluation context (time basis, trust zone, substrate class) so it can do validity-aware and revocation-aware filtering.

An illustrative interface sketch (not a library, and faithful to the disclosed ordering):

```
# Illustrative only. You implement each step.
def gate(action, agent):
    aliases   = required_aliases(action, agent)     # which policies govern
    candidates = resolve(aliases, context(agent))   # Dynamic Alias System r
    fresh     = filter_freshness(candidates, now(), agent.checkpoints)
                  #  -> validity window, revocation, anti-rollback BEFORE veri
    verified  = [p for p in fresh if verify_signature(p)]
    if not authorizes(verified, aliases, action):    # scope + freshness rec
        return DENY(action)                          # valid non-execution o
    return PERMIT(action)                            # only now may an execu
```

5. **Implement freshness filtering as a distinct stage that runs before verification.** Validity-window filter, then revocation check against resolved revocation artifacts, then anti-rollback (compare version indicators to a monotonicity constraint and validate required continuity references). Keep this ahead of signature verification so ineligible candidates are dropped deterministically.
6. **Give each agent a latest-known-good checkpoint in its own memory.** After a successful authorization, record the fingerprint or version of the accepted policy in the agent's embedded memory (and in the audit ledger). On the next evaluation, reject any candidate that would roll back below that checkpoint. This is precisely what makes downgrade rejection work while offline.
7. **Wire the gate as a hard precondition, with denial as a real outcome.** The gate must be the only path to instantiation for governed action classes, with no fallback that executes on gate failure. Return a structured denial result (action identifier, evaluated policy fingerprints, and a failure-basis indicator such as validity-window, revocation, or anti-rollback failure) and store it in agent memory and the audit ledger.

8. **Apply the same gate to mutation, delegation, and propagation.** Execution is not the only governed action. Treat "modify my policy reference field," "spawn a descendant," and "migrate to another substrate" as governed transitions that pass the same pipeline. This is what closes the policy-stripping loophole: removing a required reference is itself a mutation the gate can deny.
9. **Design supersession as quorum override, not edit.** Build the override path so a replacement object requires co-signatures meeting a threshold plus a continuity reference to the superseded object, published under the same alias. Make the gate verify quorum satisfaction and continuity at runtime before honoring an override.
10. **Stand up the append-only audit ledger with an integrity chain.** Link entries cryptographically so tampering is detectable, record every governance-relevant event, and keep the ledger off the critical authorization path so gating does not depend on logging success.
11. **Add offline revalidation rules and, if useful, fallback monitors.** Specify exactly when cached authority may be reused and when it must be denied for want of revalidation. Optionally add out-of-band fallback enforcement agents for cross-substrate consistency checks as defense-in-depth.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install and nothing here "just works" out of the box. You implement the resolver, the gate, the policy schema, the freshness and anti-rollback logic, the quorum override path, and the audit ledger yourself, and you make the engineering choices the filing leaves to embodiment (which signature scheme, which storage substrate, which dissemination network, which revocation representation).

It is a disclosed architecture, not a benchmarked or productized system. The filing does not state performance numbers, throughput, or latency, and this guide does not either. Treat any figures you need as things you must measure in your own build.

The design governs whether a prohibited action class can be instantiated; it is not an alignment system and does not judge intent or predict outcomes. It presumes your policy body actually expresses the constraints you care about, that your trust model and key or continuity management are sound, and that agents cannot forge the verification material. The audit ledger gives tamper-evident evidence; it does not by itself prevent execution, and the filing is explicit that gating does not depend on it. The approach fits autonomous, distributed, stateful, or self-modifying agents; if your agents are fully centralized and always online, a conventional trusted-runtime control may be simpler and sufficient.

Disclosure Scope

The architecture described in this guide is disclosed in United States Patent Application 19/561,229, "Cryptographically Enforced Governance for Autonomous Agents and Distributed Execution Environments," and reflects its home inventive step, the Cryptographic Governance inventive step. This guide is educational. It explains an approach so that a skilled developer can understand and build it themselves. It is not a warranty, a specification, or an offer of software, and it does not grant any license. Every statement here about how the approach works is drawn from the filing; where the filing leaves a mechanism to implementation, this guide says so rather than inventing detail.

PRIMARY TECHNICAL DISCLOSURE

- Ethical Enforcement as Infrastructure: Cryptographic Governance for Autonomous Systems (/articles/ethical-enforcement-as-infrastructure-cryptographic-governance-for-autonomous-systems)

SECONDARY TECHNICAL

- Governance Gate as Deterministic Precondition: No Verification, No Execution (/articles/cryptographic-governance/governance-gate)
- Canonical Alias to External Policy Indirection: Policy Evolution Without Agent Mutation (/articles/cryptographic-governance/policy-indirection)
- Immutable-by-Default Policy Objects: Governance Changes Through Successor Issuance (/articles/cryptographic-governance/immutable-policies)
- Runtime Policy Resolution Pipeline: Mandatory Verification Before Every Execution (/articles/cryptographic-governance/policy-resolution)
- Freshness, Revocation, and Anti-Rollback Controls: Preventing Stale Authority (/articles/cryptographic-governance/freshness-revocation)
- Memory-Derived Eligibility Conditioning: Past Violations Constrain Future Authorization (/articles/cryptographic-governance/memory-eligibility)
- Intent-Independent Authorization: Governance Without Alignment Scoring (/articles/cryptographic-governance/intent-independent-auth)
- Execution Feedback as Enforcement Signals: Operational Outcomes Shaping Future Authorization (/articles/cryptographic-governance/enforcement-feedback)
- Trust Degradation as State Transition: Policy-Defined Narrowing of Permitted Actions (/articles/cryptographic-governance/trust-degradation)
- Structural Quarantine: Execution Prevention Until Authorized Remediation (/articles/cryptographic-governance/structural-quarantine)
- Lineage-Constrained Governance Inheritance: Constraints That Persist Across Generations (/articles/cryptographic-governance/governance-inheritance)
- Unauthorized Fork Prevention: Lineage Continuity as Anti-Cloning Mechanism (/articles/cryptographic-governance/fork-prevention)
- Meta-Policy Objects: Higher-Order Constraints Across System Behavior Categories (/articles/cryptographic-governance/meta-policy)
- Quorum-Based Governance Override: Multi-Party Approval With Signature-Chain Continuity (/articles/cryptographic-governance/quorum-override)
- Distributed Alias Publication: Policy Dissemination Through Federated Registries (/articles/cryptographic-governance/alias-publication)
- Fallback Enforcement Agents: Distributed Monitors as Defense-in-Depth (/articles/cryptographic-governance/fallback-enforcement)

- Append-Only Governance Audit Ledger: Tamper-Evident Records of Every Authorization (/articles/cryptographic-governance/audit-ledger)
- Governance Without Persistent Keypairs: Trust-Slope Authorization Replacing Static Keys (/articles/cryptographic-governance/keyless-governance)
- Execution Eligibility Indicator: Dynamic Computation From Policy, Memory, and Lineage (/articles/cryptographic-governance/eligibility-indicator)
- Cross-Domain Spatial-Temporal Escalation (/articles/cryptographic-governance/cross-domain-spatial-temporal-escalation)
- Cross-Authority Handoff Governance (/articles/cryptographic-governance/cross-authority-handoff-governance)
- The Guardrail an Agent Can't Remove: Gating an Agent's Mutation of Its Own Policy, Role, Memory, and Lineage (/articles/cryptographic-governance/self-modification-governance)

APPLICATIONS · GENERAL

- Cryptographically Enforced Governance for SCADA and OT: Gating Autonomous Control Actions in Power, Water, and Industrial Control Systems (/articles/cryptographic-governance/critical-infrastructure-ics)
- How to Make High-Risk AI Agents EU AI Act Compliant by Architecture (/articles/cryptographic-governance/eu-ai-compliance)
- Self-Verifying Financial Audit Trails Without Trusted Intermediaries (/articles/cryptographic-governance/financial-audit-trails)
- Enforcing HIPAA at Every Data Operation: Structural Healthcare Compliance (/articles/cryptographic-governance/healthcare-compliance)
- Preventing Classified Data Spillage: Cryptographic Classification Enforcement for Defense (/articles/cryptographic-governance/defense-classification)
- Tamper-Evident Environmental Monitoring: Cryptographic Governance for Emissions and Compliance Data (/articles/cryptographic-governance/environmental-monitoring)
- Pharmaceutical Supply Chain Governance: DSCSA, FMD, and Cold-Chain Compliance Bound to the Product (/articles/cryptographic-governance/pharmaceutical-supply)
- Cryptographic Governance for Nuclear Facility Operations: Structural Enforcement of Technical Specifications (/articles/cryptographic-governance/nuclear-facility-governance)
- Preventing CSAM Distribution at the Source: Cryptographic Governance for Child Safety Content Enforcement (/articles/cryptographic-governance/child-safety-enforcement)
- Coalition Policy Distribution Without Shared Authority (/articles/cryptographic-governance/coalition-policy-distribution)
- EU AI Act Recital 73 and Article 14: How to Build AI That Cannot Disable Its Own Oversight (/articles/cryptographic-governance/eu-ai-act-self-constraint)

- Enforcing Build Provenance Before Artifacts Ship: Cryptographic Governance for Software Supply-Chain Integrity (/articles/cryptographic-governance/software-supply-chain-provenance)

APPLICATIONS · SPECIFIC

- HashiCorp Vault Alternative for Governed Agent Execution: Binding Policy to Action (/articles/cryptographic-governance/hashicorp-vault)
- AWS KMS Manages Encryption Keys. The Keys Do Not Carry Governance. (/articles/cryptographic-governance/aws-kms)
- Open Policy Agent Decoupled Policy From Code. The Policy Is Not Cryptographically Bound. (/articles/cryptographic-governance/open-policy-agent)
- Styra vs Cryptographically Governed Agent Execution: Beyond Advisory Policy (/articles/cryptographic-governance/styra)
- Snyk vs Cryptographic Governance: Vulnerability Scanning Is Not Runtime Enforcement (/articles/cryptographic-governance/snyk)
- Palo Alto Networks Inspects Traffic. It Does Not Govern the Operations That Generate It. (/articles/cryptographic-governance/palo-alto)
- SPIFFE/SPIRE vs Governed Agent Execution: Workload Identity Without a Cryptographic Policy Binding (/articles/cryptographic-governance/spiffe-spire)
- cert-manager vs Cryptographic Governance: Certificates Authenticate Identity, They Do Not Gate Execution (/articles/cryptographic-governance/cert-manager)
- Keycloak vs Cryptographically Governed Agent Execution: Beyond Identity Tokens (/articles/cryptographic-governance/keycloak)
- HashiCorp Boundary Alternative for Governed Session Operations: Zero-Trust Access vs Cryptographic Governance (/articles/cryptographic-governance/boundary)
- Teleport Alternative for Governed Operations: Access Control Is Not Cryptographic Governance (/articles/cryptographic-governance/teleport)
- BeyondTrust vs Cryptographic Governance: PAM Manages Privilege, It Does Not Bind Operations to Signed Policy (/articles/cryptographic-governance/beyondtrust)
- CyberArk vs Cryptographically Governed Agent Execution: PAM Protects the Credential, Not the Operation (/articles/cryptographic-governance/cyberark)
- 1Password vs Cryptographically Governed Agent Execution: Credential Custody Is Not Bound Governance (/articles/cryptographic-governance/1password)
- The Update Framework (TUF) / Notary alternative: signing software artifacts vs governing what an agent may do at runtime (/articles/cryptographic-governance/tuf-notary)
- Sigstore (cosign / Rekor) alternative: enforcing signed policy before an autonomous agent acts (/articles/cryptographic-governance/sigstore)

Cryptographic Governance overview → (/cryptographic-governance)