

Verifiable Delegated Authority for LLM Agents in Banking: A Formal Architecture, Capability Protocol, and Audit Guarantees

Neeraj Aggarwal

neeraj.trac3@gmail.com

Infosys Limited

Research Article

Keywords: agentic AI, delegation, capability security, multi-agent verification, verifiable workflows, banking

Posted Date: October 20th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11415811/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Verifiable Delegated Authority for LLM Agents in Banking: A Formal Architecture, Capability Protocol, and Audit Guarantees

Neeraj Aggarwal
Infosys Limited, USA
ORCID: 0009-0006-3854-9885

Abstract

Large language model (LLM) agents are beginning to act in banking infrastructure, reconciling payments, triaging fraud alerts and orchestrating back-office workflows. Because a non-deterministic reasoner initiates authenticated, high-value actions, the actor assumptions of existing API security models no longer hold. Building on recent work that moves authorization out of the model and into the execution boundary, we give a formal treatment tailored to regulated banking. An agent is modelled as an untrusted stochastic proposer of typed actions whose authority is a capability set attenuated along a delegation chain, mediated by a deterministic policy engine and single-use, parameter-bounded tokens. We prove (i) authority containment under an arbitrary adversarial proposer, (ii) a closed-form bound on unattended financial loss, (iii) a common-cause analysis of multi-agent review showing that reviewer diversity, not reviewer count, sets the residual-risk floor, with a cost-aware invocation rule, and (iv) soundness of an evidence-only verifier over signed, hash-chained logs, including resistance to log rollback; we also bound the sample size needed to detect policy drift. A reference implementation and simulation with an adversarial proposer (no LLM in the loop) found no unauthorized execution in 700,000 attempts, showed that the loss bound is tight, and exposed two protocol gaps (token expiry and log rollback) that the final design closes. LLM-in-the-loop evaluation and mechanized proofs remain future work.

Keywords: agentic AI, delegation, capability security, multi-agent verification, verifiable workflows, banking

1 Introduction

Banks have spent two decades hardening their internet-facing surface: TLS everywhere, OAuth-scoped APIs, hardware security modules and layered transaction fraud detection. This hardening assumes a stable actor model, namely a human in an authenticated session or a deterministic service calling a versioned API. An LLM agent violates both assumptions. It reads unstructured context (emails, tickets, reconciliation reports), reasons over it, and selects a sequence of actions that may differ across runs on the same input. This is what makes agents useful for judgment-heavy back-office work, and also what makes them hazardous when connected to systems that move money.

The multi-agent systems literature has long treated agents as entities with delegated goals and bounded autonomy [33]. Banking adds a requirement that this literature seldom states formally: a third party (an auditor, a regulator, or another agent) must be able to *check* that an agent stayed within its delegated authority, without trusting the agent’s self-report. Early work on LLM agent safety concentrated on detecting or filtering prompt injection at the model level [23, 16, 9]. A growing line of work instead moves enforcement out of the model and into the execution boundary, using capability-based control and per-action

authorization [4, 2, 1]. We adopt that position and do not claim it as new. We ask what further guarantees a regulated banking setting requires: a bound on financial loss, a quantitative account of multi-agent review, and evidence that a third party can check without access to the model (Section 2 compares our scope with this work).

Contributions.

1. A formal model of delegated authority for LLM agents in banking (Section 3): typed financial actions, capability sets, attenuating delegation chains, and a deterministic attribute-based policy engine (ABAC) with a human-approval lifecycle.
2. A protocol of signed intent envelopes and single-use, parameter-bounded capability tokens bound to chain expiry (Section 5), with proofs of *authority containment* (Theorem 1) and a *tight bound on unattended loss* (Theorem 2).
3. A common-cause model of multi-agent review (Section 6) that quantifies the value of reviewer diversity, gives an upper bound on the common-cause parameter from observable joint failures, and yields a cost-aware review-invocation rule.
4. A definition of *verifiable workflow*, a tamper-evidence theorem that includes rollback resistance via an independent witness, and a soundness theorem for an evidence-only verifier (Section 7), plus a sample-complexity bound for policy-drift detection.
5. A reference implementation and simulation under an adversarial proposer (Section 9) that tests containment, loss-bound tightness, tamper evidence and overhead, and that exposed two protocol gaps which the final design closes.

We use “agentic” rather than “autonomous” throughout: every result assumes the agent acts inside an explicit, enforced boundary of delegated authority.

2 Related Work

Capability security. Capabilities originate in the work of Dennis and Van Horn [12] and are surveyed by Levy [19]. Macaroons [8] introduced attenuable bearer credentials with contextual caveats; our delegation chains use the same attenuation principle but with typed, parameter-level scopes. SDSI/SPKI [26] treated delegation of authority in decentralized settings.

Enforceable policy and access control. Schneider [28] characterizes the policies enforceable by execution monitors, which is the class our policy engine falls into. ABAC is specified in NIST SP 800-162 [18]. Zero-trust architecture [27] and workload identity (SPIFFE/SPIRE [29]) supply the identity substrate. Token delegation follows OAuth 2.0 [24] and token exchange [25].

Tamper-evident evidence. Hash-chained and Merkle-structured logs [20, 11] and software supply-chain attestation systems [31, 21] underlie our evidence layer.

LLM agent safety. Prompt injection [23, 16], adversarial alignment [9] and unfaithful chain-of-thought explanations [32] motivate treating the model as untrusted and its narrative as non-evidentiary. Chan et al. [10] discuss visibility into AI agents at a governance level; we provide a concrete formal evidence model.

LLM-agent authorization and enforcement. Three recent systems are closest to ours. CaMeL [4] wraps an untrusted LLM in a capability-based interpreter that tracks data flow and applies policies before tool execution. AgentGate [2] mediates every tool invocation through a runtime authorization layer, issues short-lived budget-bounded capability tokens, formalizes properties such as delegation non-escalation and containment in Dafny, and evaluates on AgentDojo [5] and InjecAgent. ActGov [1] enforces policies per action and verifies them with SMT-based checking. Our authorization core (typed intents, capability tokens, non-escalating delegation) is in the same spirit and is not offered as a new idea. Based on the published descriptions we reviewed, our scope differs in four respects: (i) a banking-specific action and policy model with an explicit approval lifecycle and a closed-form bound on unattended *financial* loss; (ii) a quantitative common-cause analysis of multi-agent review; (iii) an evidence-only verifier with a tamper-evidence theorem that includes log rollback; and (iv) a sample-complexity bound for policy drift across model versions. Conversely, those systems offer mechanized proofs and benchmark evaluation with real LLMs, which we do not (Section 10).

Delegation, institutions and verification in multi-agent systems. The multi-agent systems literature has long studied delegation of tasks and authority [3], norm-governed agent interaction through electronic institutions [6], and verification of autonomous systems [7]. Our delegation chains and approval lifecycle can be read as a minimal institution for LLM agents in which the execution layer plays the role of the institutional enforcer, and our evidence-only verifier as a runtime-verification oracle that needs no access to agent internals.

Regulatory context. Model-risk expectations [30], the NIST AI RMF [22], the EU AI Act [14] and DORA [13] frame the supervisory setting for AI-driven decisioning in banking.

3 System Model and Threat Model

3.1 Actions, authority and delegation

Let $\mathcal{T}$ be a finite set of *action types*. Each $\tau \in \mathcal{T}$ has a parameter space θ_τ (e.g., amount, currency, counterparty, case reference). The *action space* is $\mathcal{A} = \{(\tau, \theta) : \tau \in \mathcal{T}, \theta \in \theta_\tau\}$. Each action has a non-negative *loss exposure* $L : \mathcal{A} \rightarrow \mathbb{R}_{\geq 0}$ (typically the monetary amount at risk).

A *capability set* is any $\Gamma \subseteq \mathcal{A}$. A *grant* is a tuple $g = (p, q, \Gamma, t_{\text{exp}})$ in which principal p delegates capability set Γ to principal q until time t_{exp} . Principals are humans, business processes, services or agent instances.

Definition 1 (Valid delegation chain). A sequence $d = (g_1, \dots, g_k)$, $g_i = (p_{i-1}, p_i, \Gamma_i, t_i)$, is valid at time t if (a) p_0 is a registered root principal (a human or business process holding authority for Γ_1), (b) $\Gamma_{i+1} \subseteq \Gamma_i$ and $t_{i+1} \leq t_i$ for all $i < k$ (attenuation), and (c) $t \leq t_k$. Its effective authority is $\text{Auth}(d) = \Gamma_k$.

Lemma 1 (No privilege escalation). For any valid chain $d = (g_1, \dots, g_k)$ and any prefix $d_j = (g_1, \dots, g_j)$, $\text{Auth}(d) \subseteq \text{Auth}(d_j) \subseteq \Gamma_1$. Consequently, in any orchestration tree in which each callee’s chain extends its caller’s, every descendant’s authority is contained in the root’s grant.

Proof. Induction on k using $\Gamma_{i+1} \subseteq \Gamma_i$. The orchestration-tree claim follows because each edge extends the chain by one attenuating grant. $\square$

Lemma 1 formalizes the *privilege creep* threat: a low-privilege triage agent cannot transitively reach a high-privilege action, because no valid chain through it contains that action.

3.2 Agents as untrusted stochastic proposers

For a context space $\mathcal{C}$ (the agent’s observed inputs), an agent with configuration m (model family, version, system-prompt hash) induces a kernel $\pi_m(\cdot | c)$ over *proposed intents* in $\mathcal{A} \cup \{\perp\}$. We make no assumption on π_m . In particular, in the threat model below it may be replaced by an arbitrary, adaptively chosen kernel.

Trusted computing base (TCB). The TCB comprises: the policy engine and its signing key sk_{pol} ; the execution layer and its spent-token set; the append-only log and its signing key; and the key infrastructure for principals’ identities.

Adversary. The adversary controls the agent’s context c (e.g., by injecting instructions in documents) and the proposer kernel, observes all public messages, and may replay any message. It does not hold keys in the TCB, and cannot violate the integrity of the TCB. Signature schemes are assumed existentially unforgeable under chosen-message attack (EUF-CMA) [15], and the hash H collision-resistant. Reasoning opacity is handled by construction: no result below depends on the model’s stated reasoning, consistent with evidence that it can be unfaithful [32].

3.3 Design requirements

The model supports four requirements: **R1** every action is attributable to an identity and a delegation chain; **R2** authority is capability-scoped and time-boxed; **R3** the execution layer, not the model, enforces authority (the model proposes, the policy engine disposes); **R4** every step emits independently checkable evidence.

Notation.

Symbol	Meaning
$\mathcal{A}, \theta_\tau$	typed action space; parameter space of type τ
$L(a)$	loss exposure of action a
$d, \text{Auth}(d)$	delegation chain; its effective authority

Symbol	Meaning
P_v	deterministic ABAC policy, version v
x	attribute snapshot (committed by hash $H(x)$)
κ	capability token
$= (\tau, B, jti, t_{\text{nb}}, t_{\text{exp}}, H(\varepsilon))$	
m_τ, n_τ, w_τ	per-action cap, rate limit, window
q_i, γ	independent failure rate of agent i ; common-cause probability

4 Reference Architecture

The architecture has four layers, each with a separate trust boundary (Fig. 1).

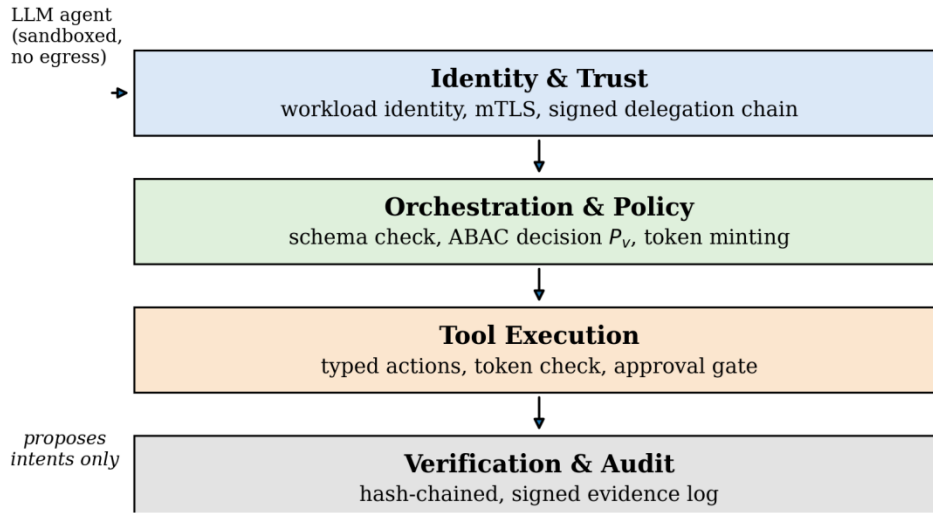


Figure 1. Four-layer reference architecture. The LLM runs in a sandbox without network egress and appears only as a proposer of intents to the orchestration layer.

Identity and trust. Each agent instance receives a workload identity (SPIFFE/SPIRE style [29]) extended with (i) a *model provenance record*: model family, version and a hash of the system prompt and configuration, and (ii) the signed delegation chain d of Section 3. An identity is the pairing of a configuration with an operating context, not “the model”. Mutual TLS terminates here for every agent-to-service call.

Orchestration and policy. The sandboxed model can only emit schema-validated intents. A deterministic policy engine evaluates each intent. We use ABAC [18] because the relevant question is rarely “may this agent call the payments API” but “may this agent, under this delegation, initiate a transfer of this size to this counterparty in this case context”. Policies are versioned and change-controlled independently of models.

Tool execution. Only this layer holds live credentials to ledgers, payment rails and case systems. It exposes typed actions (e.g., `initiate_transfer(amount,currency,counterparty,case)`) rather than a general API. Actions above risk thresholds enter a human-approval state machine: *proposed* $\rightarrow$ *policy-checked* $\rightarrow$ *pending-approval* $\rightarrow$ *executed/rejected*.

Verification and audit. Every state transition emits a signed event to an append-only, hash-chained log that references the policy version and identity credentials in force and is queryable independently of the agent and the model provider.

5 Protocols and Authorization Guarantees

5.1 Signed intent envelopes

A proposed action is wrapped in an *intent envelope*

$$\varepsilon = (a, id, d, case, nonce, t), \quad \sigma = \text{Sig}_{sk_{id}}(H(\varepsilon)),$$

where $a \in \mathcal{A}$ is the typed action, id the issuing agent's identity credential, d the delegation chain, $case$ the business case or ticket reference, and $nonce$ a fresh value. The orchestration layer rejects any envelope failing schema validation or $\text{Ver}_{pk_{id}}(H(\varepsilon), \sigma)$ before it reaches the policy engine.

5.2 Policy decision

Let $P_v: \mathcal{A} \times \mathcal{X} \rightarrow \{\text{permit}, \text{escalate}, \text{deny}\}$ be a *deterministic* policy of version v over an attribute space $\mathcal{X}$, and let x be the attribute snapshot (counterparty on-file data, secondary-source values, case history) at decision time. For an envelope ε with chain d valid at t , define

$$\text{Dec}(\varepsilon, x) = \begin{cases} \text{deny} & a \notin \text{Auth}(d) \text{ or } P_v(a, x) = \text{deny}, \\ \text{escalate} & a \in \text{Auth}(d), \text{ and } (P_v(a, x) = \text{escalate or } L(a) > m_\tau), \\ \text{permit} & \text{otherwise.} \end{cases}$$

5.3 Capability tokens

On PERMIT, the policy engine mints a token

$$\kappa = (\tau, B, jti, t_{\text{nbef}}, t_{\text{exp}}, H(\varepsilon)), \quad \sigma_\kappa = \text{Sig}_{sk_{\text{pol}}}(\kappa),$$

with $B \subseteq \{\theta \in \Theta_\tau: L(\tau, \theta) \leq m_\tau\}$ a *parameter bound* (in the simplest case a singleton), jti a unique identifier, and $[t_{\text{nbef}}, t_{\text{exp}}]$ a short validity window. The policy engine sets $t_{\text{exp}} \leq \min(t_{\text{nbef}} + \Delta, t_k)$, where Δ is the maximum token lifetime and t_k is the expiry of the last grant of the chain, so that a token can never outlive the authority it was minted under (Section 9 shows that omitting this rule permits out-of-authority execution). This follows the OAuth 2.0 token-exchange pattern [25] but with scopes at the granularity of individual

typed actions and their parameter bounds. The execution layer, on receiving $(a, \kappa, \sigma_\kappa)$ with $a = (\tau', \theta)$, runs the following check, where $\mathcal{S}$ is the set of spent identifiers:

$$\text{Exec}(a, \kappa) \Leftrightarrow \text{Ver}_{pk_{\text{pol}}}(\kappa, \sigma_\kappa) \wedge \tau' = \tau \wedge \theta \in B \wedge t \in [t_{\text{nbf}}, t_{\text{exp}}] \wedge jti \notin \mathcal{S},$$

and if it holds, executes a and sets $\mathcal{S} \leftarrow \mathcal{S} \cup \{jti\}$. The execution layer additionally enforces, per type τ , a sliding-window rate limit: at most n_τ automatic executions in any interval of length w_τ .

5.4 Authority containment

Theorem 1 (Authority containment). *Let the adversary control the proposer kernel and context (Section 3). Then, except with probability at most $\text{Adv}_{\text{Sig}}^{\text{euf-cma}}$ (negligible in the security parameter), every action a executed by the execution layer satisfies:*

1. $a \in \text{Auth}(d) \subseteq \Gamma_1$ for a chain d valid at execution time and rooted at a registered principal;
2. either $\text{Dec}(\varepsilon, x) = \text{permit}$ for the envelope ε bound in the token, or a signed human-approval record for $H(\varepsilon)$ exists;
3. $L(a) \leq m_\tau$ unless a was human-approved.

Proof. Suppose an adversary $\mathcal{B}$ causes an execution violating (i)–(iii) with non-negligible probability. We build a forger $\mathcal{F}$ against Sig that is given the verification key pk_{pol} and a signing oracle. $\mathcal{F}$ simulates the whole system for $\mathcal{B}$; whenever the simulated policy engine would mint a token, $\mathcal{F}$ obtains its signature from the oracle (the engine’s decision is a deterministic function of the envelope and snapshot, so $\mathcal{F}$ can compute it). Consider an execution of (a, κ) accepted by the execution check. There are two cases. (1) κ was minted by the simulated engine. Minting happens only after the engine has verified the envelope signature and the chain d (so $a \in \text{Auth}(d) \subseteq \Gamma_1$ by Lemma 1, and $t \leq t_k$ because $t_{\text{exp}} \leq t_k$), and after $\text{Dec} = \text{permit}$ or a signed approval; B and the check $\theta \in B$ give (iii). So (i)–(iii) hold, a contradiction. (2) κ was never submitted to the oracle. Then (κ, σ_κ) is a valid signature on a fresh message and $\mathcal{F}$ outputs it as a forgery. Hence $\Pr[\text{violation}] \leq \text{Adv}_{\text{Sig}}^{\text{euf-cma}}$. $\square$

Corollary 1 (Prompt-injection containment). *For any context c , including adversarially injected content, the set of effects reachable through the automatic path is contained in $\{a \in \text{Auth}(d) : L(a) \leq m_\tau\} \cap \{a : P_v(a, x) = \text{permit}\}$. Prompt injection can change which permitted action is proposed, never whether an unpermitted action is executed.*

Remark 1. Corollary 1 does not claim that injected instructions are harmless. An injected context may steer the agent toward a permitted but undesirable action, for example resubmitting the wrong payment within limits. The next theorem bounds such damage quantitatively. Consensus (Section 6) reduces its likelihood.

Theorem 2 (Bounded unattended loss). Let $\Lambda(T)$ denote the total L -exposure executed on the automatic path (without human approval) in any half-open time interval $(a, a + T]$. Under the per-type cap m_τ and sliding-window rate limit (n_τ, w_τ) , $\Lambda(T) \leq \sum_{\tau \in \mathcal{T}} n_\tau \lceil \frac{T}{w_\tau} \rceil m_\tau$.

Proof. A half-open interval of length T is covered by $\lceil T/w_\tau \rceil$ half-open intervals of length w_τ , each containing at most n_τ automatic executions of type τ , each of exposure at most m_τ by Theorem 1(iii). Summing over types gives the bound. $\square$

Corollary 2 (Token leakage). A token κ leaked to an adversary permits at most one execution, within B and before t_{exp} , hence additional loss at most $\max_{\theta \in B} L(\tau, \theta) \leq m_\tau$. A role-scoped bearer token, in contrast, permits unboundedly many calls over all of Θ_τ until revoked.

Figure 2 plots the bound of Theorem 2 for a single action type under three hypothetical rate limits. The bound grows linearly in the observation window and is set entirely by configuration, not by model behavior. The markers show the maximum exposure observed in the simulation of Section 9, where an adversary saturates the rate limit; they meet the bound with equality, so the bound is tight.

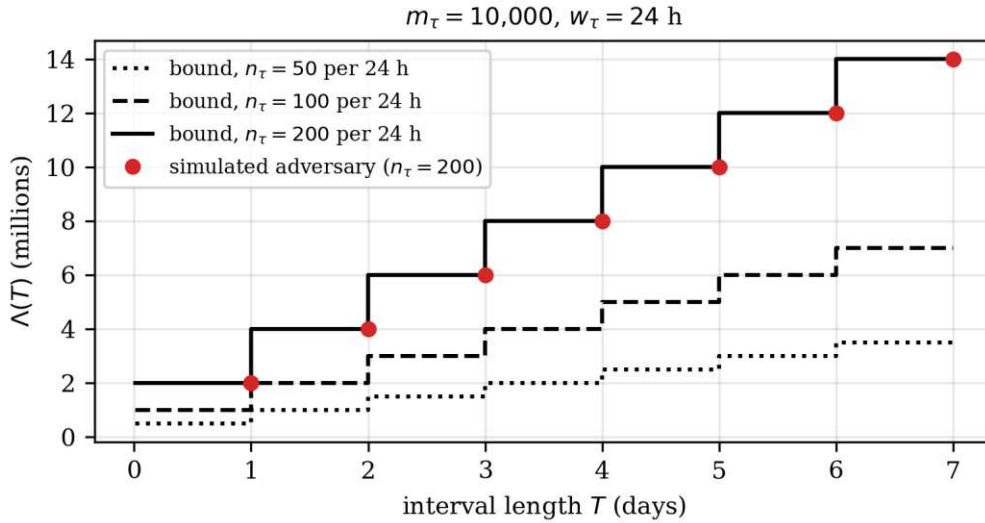


Figure 2. Bound on unattended loss $\Lambda(T)$ from Theorem 2 (lines), for one action type with $m_\tau = 10,000$ and $w_\tau = 24$ h and three rate limits n_τ , and the maximum exposure observed under a saturating adversary in the simulation of Section 9 (markers, $n_\tau = 200$). Parameters are illustrative and the data are synthetic.

6 Multi-Agent Consensus

For high-risk intents the architecture adds a second, independently configured *reviewer* agent that concurs or flags the envelope; disagreement routes to a human. We analyze the

residual risk. Fix an intent that is in truth harmful. Let F_i be the event that agent $i \in \{0, 1, \dots, k\}$ (agent 0 the proposer, the rest reviewers) fails, i.e., proposes or concurs with it.

Common-cause model. In the *beta-factor* style of common-cause failure modelling used in reliability engineering, with probability $\gamma \in [0, 1]$ a common-cause event occurs (e.g., an injected payload effective against all agents, or a shared training-data blind spot) and all agents fail; otherwise agents fail independently with probabilities q_i .

Proposition 1 (Residual risk of consensus). *Under the common-cause model, $\Pr\left[\bigcap_{i=0}^k F_i\right] = \gamma + (1 - \gamma) \prod_{i=0}^k q_i \geq \gamma$, whereas the marginal failure probability of a single agent is $p_i = \gamma + (1 - \gamma)q_i$. Hence the reduction obtainable from adding reviewers is bounded by $p_0 - \gamma$, no matter how many reviewers are added.*

Proof. Condition on the common-cause event C : $\Pr[\bigcap F_i \mid C] = 1$ and, by independence, $\Pr[\bigcap F_i \mid \neg C] = \prod q_i$. The law of total probability gives the first expression. The marginal follows identically. $\square$

Remark 2 (Illustration with hypothetical parameters). Take $q_0 = q_1 = 0.02$. With $\gamma = 0.05$, one agent fails with probability $p = 0.05 + 0.95 \cdot 0.02 = 0.069$, and two-agent consensus with $0.05 + 0.95 \cdot 0.0004 \approx 0.0504$, a reduction of only about 27%. With $\gamma = 0.005$ (a diverse reviewer from a different model family with a different prompt and tooling), the figures are 0.0249 and ≈ 0.0054 , a reduction of about 78%. This is the quantitative argument for reviewer *diversity* over reviewer *count*. The values of γ here are illustrative; estimating γ empirically is an open measurement problem (Section 9).

Remark 3 (Bounding γ from data). The parameter γ is not directly observable, and the model assumes at most one common-cause event per harmful intent. However, Proposition 1 gives $\Pr[\bigcap_i F_i] \geq \gamma$, so the empirical joint failure rate of a reviewer set on a labelled set of harmful intents, with a Hoeffding confidence margin, yields a usable *upper bound* on γ . This is the quantity an institution would measure to decide whether a candidate reviewer is diverse enough.

Figure 3 visualizes Proposition 1: panel (a) shows residual risk converging to the floor γ as reviewers are added, and panel (b) shows that additional reviewers beyond the first or second yield negligible benefit unless γ is reduced.

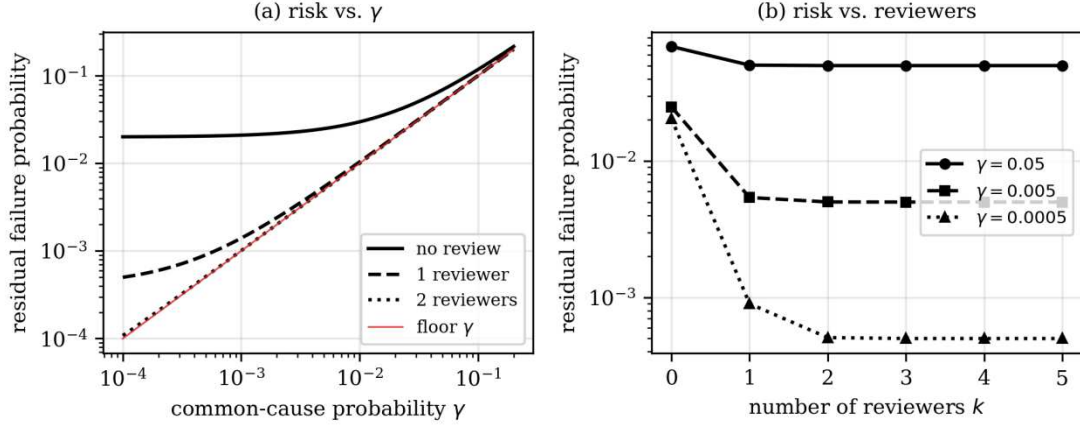


Figure 3. Residual failure probability under the common-cause model of Proposition 1 with independent failure rate $q_i = 0.02$. (a) Risk against the common-cause probability γ for zero, one and two reviewers; the thin line is the floor γ . (b) Risk against the number of reviewers k for three values of γ . Analytical curves; parameters are illustrative.

When to invoke review. Let $h(a)$ be the probability that the proposed intent a is harmful, given the features available to the policy engine, and δ_c the conditional probability that the reviewer concurs given that the proposer failed (so $\delta_c = \Pr[F_1 | F_0]$). Let c_r be the expected cost (compute, latency, delay penalty) of one review.

Proposition 2 (Cost-aware review rule). *Invoking review reduces expected loss by $h(a)L(a)(1 - \delta_c)$, so review should be invoked if and only if $h(a)L(a)(1 - \delta_c) > c_r$. Under the common-cause model, $\delta_c = \frac{\gamma + (1-\gamma)q_0q_1}{\gamma + (1-\gamma)q_0}$.*

Proof. Without review the expected loss from a harmful proposal is $h(a)L(a)$. With review, a loss occurs only if the reviewer also concurs, with conditional probability δ_c , giving $h(a)L(a)\delta_c$. The difference is the stated reduction, and review is worthwhile iff it exceeds c_r . The expression for δ_c is $\Pr[F_0 \cap F_1] / \Pr[F_0]$ from Proposition 1. $\square$

Since $L(a)$ is bounded by m_τ on the automatic path, Proposition 2 induces a threshold in L above which review is mandatory, consistent with tiered risk thresholds in practice.

7 Verifiable LLM Workflows

7.1 Definition

A *workflow run* produces an evidence sequence $E = (e_1, \dots, e_n)$ of events (attested snapshot, intents, policy decisions, tokens, approvals, execution results).

Definition 2 (Verifiable workflow). *A workflow architecture is verifiable if there is a deterministic algorithm $\mathcal{V}(E, PK, \{P_v\}) \in \{\text{accept}, \text{reject}\}$ that takes as input only the evidence, public keys and policy versions (not the model, its weights or its reasoning) such that*

- **completeness:** an honest run is accepted;
- **soundness:** if $\mathcal{V}$ accepts E , then (a) the commitment to the input context and attribute snapshot was logged, with a valid signature, before the first proposal event of the run, (b) each executed action a equals the action in an envelope ε with $\text{Dec}(\varepsilon, x) = \text{permit}$ or a signed approval, for the snapshot x committed in the log, and (c) the executed parameters lie in the bound B of the corresponding token.

This is deliberately weaker, and more tractable, than interpretability: it certifies *what was authorized and done*, not *why the model proposed it*.

7.2 Evidence log

Let $h_0 = H(\text{genesis})$ and $h_i = H(h_{i-1} \parallel e_i)$, where each e_i contains its policy version, identity credentials, and the hash of the attribute snapshot $H(x)$. The log periodically publishes a signed head $(n, h_n, \text{Sig}_{sk_{\log}}(n \parallel h_n))$.

Theorem 3 (Tamper evidence with rollback resistance). *Let n_w be a witness count: a lower bound on the length of the genuine log, obtained from a source outside the log operator's control (for example an external timestamping or transparency service, or a counter in a separate trust domain). A verifier accepts an evidence sequence E' with a signed head (n, h_n, σ) only if σ is valid, E' has length n and hash-chains to h_n , and $n \geq n_w$. Then, except with probability at most $\text{Adv}_H^{\text{coll}} + \text{Adv}_{\text{Sig}}^{\text{euf-cma}}$, every accepted E' agrees with the genuine log E on its first n_w entries. Without the witness check, an older validly signed head together with the corresponding log prefix is accepted (rollback), so the witness requirement cannot be dropped.*

Proof. Absent a forgery, a head with a valid σ was signed by the log operator over a prefix of the genuine log, so h_n is the chain hash of $E_{1..n}$. If E' has length n and chains to h_n but $E' \neq E_{1..n}$, let j be the largest index with $(h'_{j-1}, e'_j) \neq (h_{j-1}, e_j)$. Then $H(h'_{j-1} \parallel e'_j) = h'_j = h_j = H(h_{j-1} \parallel e_j)$ with different preimages, a collision. Hence $E' = E_{1..n}$ and, since $n \geq n_w$, E' agrees with E on the first n_w entries. The rollback claim follows by exhibiting the attack: an older head satisfies every check except $n \geq n_w$. $\square$

Theorem 4 (Soundness of the evidence-only verifier). *Let $\mathcal{V}$ (i) checks all signatures and the hash chain against a head that satisfies the witness condition of Theorem 3, (ii) recomputes $\text{Dec}(\varepsilon, x)$ from the logged envelope, the attribute snapshot committed by $H(x)$, and P_v at the logged version, (iii) checks that the token bound B and window match the executed action and execution time, (iv) checks that each human-approval event references the hash of the envelope executed, and (v) checks that the context-commitment event precedes the first proposal event of the run. Then $\mathcal{V}$ is sound in the above sense, except with probability $\leq \text{Adv}_H^{\text{coll}} + \text{Adv}_{\text{Sig}}^{\text{euf-cma}}$, provided P_v is deterministic and the attribute snapshot is committed before the decision.*

Proof. By Theorem 3 the accepted evidence is the evidence that was logged. Because P_v is deterministic and x is committed by $H(x)$, step (ii) is a pure function of verified data, so the

logged decision equals the recomputed one. The remaining clauses are direct checks of the execution check and of approval-hash equality. Clause (a) is the ordering check (v) on the verified chain: the context commitment precedes the first proposal, so it cannot have been fitted to the model’s behavior after the fact. The verifier does not check, and cannot check, that the model actually read the committed context; that lies outside an evidence-only guarantee. $\square$

No step of $\mathcal{V}$ requires replaying the LLM. Model non-determinism is confined to the proposal, which is not trusted, while every trusted transition is deterministic. This is the formal sense in which the architecture is verifiable even though the model is not reproducible.

7.3 Human approval as a verifiable state

An approval is an event $e = (\text{approver}, H(\varepsilon), t, \sigma_{\text{approver}})$. Clause (iv) of Theorem 4 ensures that the approver is shown, and approves, exactly the executed envelope, closing the audit gap of free-text chat approvals.

7.4 Policy drift across model versions

Policies and models version independently. Let D be a distribution over contexts and $\Phi(m, P) = \Pr_{c \sim D, a \sim \pi_m(\cdot|c)}[P(a, x_c) = \text{permit}]$ the *permissive rate* of policy P under model configuration m . A policy safe under m may become over-permissive under m' if $\Phi(m', P)$ shifts.

Proposition 3 (Drift detection sample complexity). *Suppose n i.i.d. contexts are replayed in a sandbox against each of m and m' and the empirical permissive rates $\hat{\Phi}, \hat{\Phi}'$ are computed. If $n \geq \frac{2\ln(4/\delta)}{\varepsilon^2}$, then with probability at least $1 - \delta$, $|(\hat{\Phi}' - \hat{\Phi}) - (\Phi(m', P) - \Phi(m, P))| \leq \varepsilon$. A shift of magnitude greater than 2ε is therefore detected with confidence $1 - \delta$.*

Proof. By Hoeffding’s inequality [17], each estimate deviates from its mean by more than $\varepsilon/2$ with probability at most $2\exp(-n\varepsilon^2/2)$. Requiring this to be at most $\delta/2$ yields $n \geq 2\ln(4/\delta)/\varepsilon^2$, and a union bound over the two estimates gives the claim. $\square$

For example, $\varepsilon = 0.02$ and $\delta = 0.05$ require $n \approx 2.2 \times 10^4$ replayed contexts per version. Figure 4 shows how the required sample size scales with the tolerance and the confidence level.

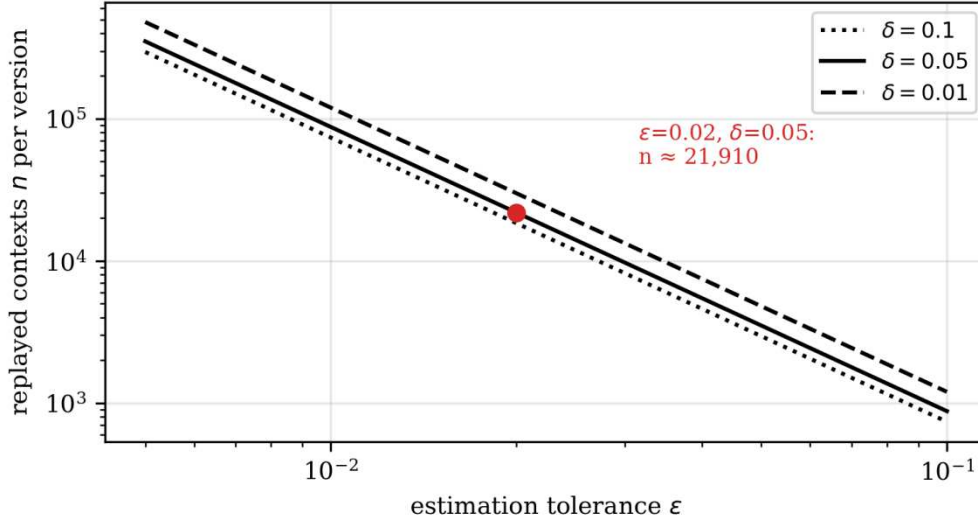


Figure 4. Sample size $n = 2\ln(4/\delta)/\varepsilon^2$ from Proposition 3 needed to estimate the shift in permissive rate between two model versions to within ε with confidence $1 - \delta$.

8 Case Study: Payment-Exception Agent

A payment fails settlement because of a counterparty account-detail mismatch. The agent should correct the detail if the correction is unambiguous and resubmit, otherwise escalate.

Instantiation. The typed action is $a = (\text{correct_and_resubmit}, \theta)$ with $\theta = (pid, f, v, case)$: payment identifier, corrected field, corrected value, case reference. Let F be an allow-list of correctable fields, $A_{\max}$ the auto-resubmission threshold, and $\sigma(\cdot)$ a verified secondary source. The policy is

$$P_v(a, x) = \text{permit} \Leftrightarrow f \in F \wedge \text{amt}(pid) \leq A_{\max} \wedge v = \sigma_x(pid, f),$$

and ESCALATE otherwise. The token has $B = \{\theta\}$ (single payment, single value), $t_{\text{exp}} - t_{\text{nbef}} = 90$ s. A reviewer agent from a different model family reviews ε (Section 6). On concurrence the action executes; on disagreement or policy failure the case goes to a specialist who sees the same structured envelope.

Guarantees (hypothetical parameters). With $m_\tau = A_{\max} = 10,000$ and $n_\tau = 200$ per $w_\tau = 24$ h, Theorem 2 bounds unattended exposure at 2×10^6 per day regardless of what the model proposes or what an injected document instructs. An auditor months later runs $\mathcal{V}$ of Theorem 4 on the log without access to the original model.

9 Reference Implementation and Simulation-Based Validation

Scope. We implemented the formal model as a reference implementation (about 300 lines of Python, supplied as supplementary material) and exercised it with an adversarial

proposer. This validates the *formal model and protocol logic* on synthetic data. It is not an evaluation of LLM agents: no language model is in the loop, HMAC-SHA256 stands in for digital signatures, and no claim is made about attack success rates against real models or about task utility. Such an evaluation, for example on AgentDojo-style benchmarks and a payment-exception task suite, is the main item of future work.

Setup. The ledger holds 5,000 synthetic payments with amounts in $\{100, \dots, 900,000\}$ and a trusted attribute store. The root grant Γ_1 covers `resubmit` and `transfer_out` up to 1,000,000. The agent holds an attenuated grant Γ_2 covering `resubmit` on the fields `acct`, `bic` and `name` up to $A_{\max} = 10,000$. The policy of Section 8 applies, with $n_t = 200$ per $w_t = 24$ h and token lifetime 90 s.

E1: Containment. An adversarial proposer cycles through nine attack classes: arbitrary intents, forged tokens, token replay, parameter tampering after minting, expired tokens, forged envelope signatures, out-of-scope action types, execution after chain expiry, and a non-attenuating chain. Over 5 seeds of 140,000 attempts (700,000 in total), the only executions were 600 per seed from permitted intents, and an independent oracle found **zero** violations of Theorem 1(i)–(iii). With the expiry rule $t_{\text{exp}} \leq t_k$ removed, 12,383 of 20,000 attempts executed after the delegating grant had expired, which is how this gap was found and why the rule is part of the protocol.

E2: Loss bound. An adversary that saturates the rate limit every day for seven days achieves observed maximum exposures that equal the bound of Theorem 2 for every $T \in \{1, \dots, 7\}$ days (from 2.0×10^6 to 1.4×10^7 ; Fig. 2), so the bound is tight.

E3: Tamper evidence. On logs of 500 events with 2,000 trials per class (Table 1), modification, deletion, insertion and reordering were detected in all trials. Rollback to an older signed head was detected in **0%** of trials without a witness and in 100% with the witness check of Theorem 3. This exposed the rollback weakness in our first formulation of the theorem.

Detection rate over 2,000 trials per attack class (log length 500).

Attack on the log	Without witness	With witness
Modify an event	100%	100%
Delete an event	100%	100%
Insert an event	100%	100%
Reorder two events	100%	100%
Roll back to an older signed head	0%	100%

E4–E6: Consistency checks and overhead. (E4) A Monte Carlo run of the common-cause model (2×10^6 trials, $q = 0.02$, $\gamma \in \{0.05, 0.005, 0.0005\}$, $k \in \{0, 1, 2\}$) matched Proposition 1 within Monte Carlo error (largest absolute deviation 1.0×10^{-4}). This checks the algebra only, since the simulation implements the model that defines the formula. (E5) For Proposition 3 with $\varepsilon = 0.02$ and $\delta = 0.05$ ($n = 21,911$), the deviation was within ε in 100% of 3,000 trials against a guarantee of 95%, as expected for a conservative Hoeffding bound. (E6) The per-action pipeline (envelope check, policy, token minting, execution,

logging) took a median of $45\ \mu\text{s}$ (99th percentile $141\ \mu\text{s}$) single-threaded in Python. This excludes network, HSM signing, human approval and LLM latency, so enforcement cost is unlikely to be the bottleneck.

Limitations of the validation. The simulation shares assumptions with the proofs (a trusted computing base, deterministic policy, honest key handling), so it tests implementation logic and protocol gaps rather than those assumptions. Real-model injection resistance, policy quality, and utility under constraint remain unevaluated.

10 Discussion and Open Challenges

Limitations. The guarantees are only as strong as the TCB and the policy: a mis-specified P_v permits harmful actions, and the theorems say nothing about the *quality* of the agent’s judgement within the permitted set. The proofs are pen-and-paper and not mechanized, unlike some concurrent systems [2]. The validation of Section 9 uses no LLM, so it does not measure injection resistance against real models or the utility cost of enforcement. The common-cause parameter γ can only be bounded from above from data, and the model assumes a single common-cause event per intent.

Open problems. (i) Cross-institution agent identity: no widely adopted standard carries model-provenance claims analogous to X.509 for workload identity. (ii) Policy-drift tooling beyond sampling-based detection (Proposition 3). (iii) Cost-optimal consensus under correlated failures at scale. (iv) Mechanized proofs and LLM-in-the-loop evaluation on injection benchmarks and a payment-exception task suite. (v) Regulatory alignment as supervisory expectations for AI-driven decisioning continue to develop [30, 22, 14, 13].

11 Conclusion

We treat the LLM as an untrusted stochastic proposer of typed, bounded actions, make the execution layer the enforcement point, and make workflows verifiable from signed, hash-chained evidence. This yields provable authority containment, a closed-form bound on unattended loss, a quantitative account of consensus diversity, and a sound evidence-only verifier, none of which depends on the model’s reasoning. Institutions that adopt this boundary can extend agentic automation more deeply into back-office and customer-facing workflows. Those that bolt agents onto existing API surfaces without it create a largely unaudited attack surface in core banking infrastructure.

Declarations

Funding: The author received no funding for this work.

Competing interests: The author is employed by Infosys Limited and reports no employer funding, support or interest in this work.

Data and code availability: The reference implementation (`sim.py`) and raw results (`sim_results.json`) behind Section 9 are provided as supplementary material. All data are

synthetic; no real customer data were used. **Use of AI tools:** Anthropic’s Claude, a large language model, was used to help restructure and draft text and formal sections, write the simulation code, generate figures and format the manuscript. The author set the research direction, reviewed, verified and edited all content, and takes full responsibility for the work. No AI tool is listed as an author. **Views expressed** are the author’s own and do not necessarily represent those of the employer.

References

- [1] “ActGov: Governing LLM agent actions via policy-constrained validation,” arXiv:2609.24446, 2026.
- [2] “AgentGate: Capability-based runtime access control for LLM agents,” in *Proc. NeurIPS*, 2026, <https://neurips.cc/virtual/2026/168219>.
- [3] C. Castelfranchi and R. Falcone, “Towards a theory of delegation for agent-based systems,” *Robot. Auton. Syst.*, vol. 24, no. 3–4, pp. 141–157, 1998.
- [4] E. Debenedetti et al., “Defeating prompt injections by design,” arXiv:2503.18813, 2025.
- [5] E. Debenedetti et al., “AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents,” in *Proc. NeurIPS Datasets and Benchmarks*, 2024.
- [6] M. Esteva, J. A. Rodríguez-Aguilar, B. Rosell, J. L. Arcos, “On the formal specification of electronic institutions,” in *Agent Mediated Electronic Commerce*, LNAI 1991, Springer, 2001.
- [7] M. Fisher, L. Dennis, M. Webster, “Verifying autonomous systems,” *Commun. ACM*, vol. 56, no. 9, pp. 84–93, 2013.
- [8] A. Birgisson, J. G. Politz, Ú. Erlingsson, A. Taly, M. Vrabie, M. Lentczner, “Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud,” in *Proc. NDSS*, 2014.
- [9] N. Carlini et al., “Are aligned neural networks adversarially aligned?” in *Proc. NeurIPS*, 2023.
- [10] A. Chan et al., “Visibility into AI agents,” in *Proc. ACM FAccT*, 2024.
- [11] S. A. Crosby and D. S. Wallach, “Efficient data structures for tamper-evident logging,” in *Proc. USENIX Security*, 2009.
- [12] J. B. Dennis and E. C. Van Horn, “Programming semantics for multiprogrammed computations,” *Commun. ACM*, vol. 9, no. 3, pp. 143–155, 1966.
- [13] European Parliament and Council, “Regulation (EU) 2022/2554 on digital operational resilience for the financial sector (DORA),” 2022.
- [14] European Parliament and Council, “Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (AI Act),” 2024.

- [15] S. Goldwasser, S. Micali, R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, 1988.
- [16] K. Greshake et al., "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *Proc. ACM AISec*, 2023.
- [17] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Statist. Assoc.*, vol. 58, no. 301, pp. 13–30, 1963.
- [18] V. C. Hu et al., "Guide to attribute based access control (ABAC) definition and considerations," NIST SP 800-162, 2014.
- [19] H. M. Levy, *Capability-Based Computer Systems*. Digital Press, 1984.
- [20] R. C. Merkle, "A digital signature based on a conventional encryption function," in *CRYPTO*, 1987.
- [21] Z. Newman, J. S. Meyers, S. Torres-Arias, "Sigstore: Software signing for everybody," in *Proc. ACM CCS*, 2022.
- [22] NIST, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, 2023.
- [23] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications," 2024.
- [24] D. Hardt, Ed., "The OAuth 2.0 authorization framework," IETF RFC 6749, 2012.
- [25] M. Jones et al., "OAuth 2.0 token exchange," IETF RFC 8693, 2020.
- [26] R. L. Rivest and B. Lampson, "SDSI—a simple distributed security infrastructure," MIT working paper, 1996.
- [27] S. Rose, O. Borchert, S. Mitchell, S. Connelly, "Zero trust architecture," NIST SP 800-207, 2020.
- [28] F. B. Schneider, "Enforceable security policies," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 1, pp. 30–50, 2000.
- [29] SPIFFE/SPIRE Project, "Secure Production Identity Framework for Everyone," Cloud Native Computing Foundation, 2024.
- [30] Board of Governors of the Federal Reserve System and OCC, "Supervisory guidance on model risk management," SR 11-7, 2011.
- [31] S. Torres-Arias, H. Wolf, A. Kuppusamy, R. Curtmola, J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *Proc. USENIX Security*, 2019.
- [32] M. Turpin, J. Michael, E. Perez, S. Bowman, "Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting," in *Proc. NeurIPS*, 2023.
- [33] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowl. Eng. Rev.*, vol. 10, no. 2, pp. 115–152, 1995.