



Intent-Bound Commit Certificates: A Formal Transaction Protocol for Safe Autonomous AI Tool Execution

Kwan Hong TAN

Singapore University of Social Sciences, Singapore

DOI: <https://doi.org/10.68050/JAMS.2026.763>

Received: 02 October 2026; Accepted: 12 October 2026; Published: 20 October 2026

ABSTRACT

Autonomous artificial intelligence (AI) agents can initiate payments, modify records, and coordinate consequential application programming interfaces (APIs). Yet approved decisions may become unsafe when the external state changes, concurrent requests race, or acknowledgments disappear. This paper develops intent-bound commit certificates (IBCC), a transaction protocol that binds an action to authenticated authority, approved intent, canonical action identity, resource and policy versions, budget, expiry, and a stable idempotency key. A guarded state machine establishes conditional authorization, freshness, budget nonnegativity, and at-most-once effect application under explicit trust and atomicity assumptions. We distinguish this theorem from endpoint-level limitations and a retry indistinguishability result. The original deterministic synthetic testbench evaluates five enforcement configurations over 50,000 fault-injected episodes and six bounded two-client schedules; the full configuration records zero unsafe episodes and completes 37,758 eligible opportunities by construction of its modeled controls. To test implementation behavior independently of the abstract simulator, a new SQLite reference adapter was exercised in 12 integration tests, including eight-way duplicate-key and shared-budget races, tampering, policy changes, and fault-triggered rollback; all 12 tests passed, with one effect in the duplicate-key race and no budget overrun in the tested shared-budget race. The findings support a concrete single-database commit-boundary design, not a claim of production-wide or cross-service correctness.

Keywords: Agentic artificial intelligence; Transaction safety; Idempotency; Formal verification; Access control; Distributed systems

INTRODUCTION

Tool-using AI changes the risk profile of language models. A wrong answer can misinform a user, whereas a wrong external action may transfer funds, disclose a record, delete an artifact, or propagate through several services. ReAct demonstrated the utility of interleaving model reasoning with actions [1], while Toolformer established a mechanism for learning API invocation behavior [2]. AgentBench subsequently broadened evaluation across interactive settings [3]. None of these contributions makes model-generated calls equivalent to authenticated, serializable transactions.

Evidence from tool-integrated agent benchmarks establishes the relevance of adversarial tool responses and indirect prompt injection [4,5]. An attacker may induce a syntactically valid command that is outside the real user's mandate. Even absent an attacker, distributed systems may lose acknowledgments, retry operations, or act on stale resource versions [6–9]. These are execution-integrity problems and cannot be solved solely by stronger natural-language instructions or improved response accuracy.

The present problem is intentionally narrower than the broad runtime assurance agenda. TAN [10] proposed policy-gated autonomy-risk scoring for enterprise agents, but a risk score does not provide an atomic transaction boundary. This paper studies the distinct question: exactly what evidence must accompany an agent-proposed side effect, and what system-level conditions are sufficient to prevent particular classes of unsafe commits? The focus is transaction semantics, not a new general-purpose prompt-injection detector.



Three research questions guide the study. RQ1 asks how to bind user intent and authorization to a particular external state and side effect. RQ2 asks which safety properties can be established formally under explicit trust and atomicity assumptions, and which cannot. RQ3 asks which fault classes remain under enforcement ablations and whether key transaction guarantees hold in a separate executable storage-adapter test.

The contributions are: (i) the intent-bound commit certificate and a guarded state-transition system; (ii) a conditional safety theorem, a replay argument, and a retry indistinguishability proposition; (iii) a fault-isolated comparative testbench with explicit denominators and confidence intervals; and (iv) a separate SQLite integration test of concurrent replay, shared budgets, tampering and rollback, with explicitly limited local-storage guarantees; and (v) a practical taxonomy of endpoint capabilities. The naming and integration are proposed here, but the constituent ideas draw on established capabilities, proof-carrying artifacts, transaction commit, and idempotent API design [6–9,11,12].

RELATED WORK AND NOVELTY BOUNDARY

Language-model tool use and security

Agent systems combine planning and model-generated function calls with external state changes [1–3]. AgentDojo provides a dynamic suite for testing agent tasks and prompt-injection defenses [4]. InjecAgent documents indirect prompt-injection susceptibility in tool-integrated agents [5]. These lines of work motivate treating model output as untrusted input at a separate execution boundary. The proposed protocol does not infer intent from a model explanation: authority comes from a trusted authorization service and an independently established user mandate.

Distributed transactions and replay safety

Sagas arrange long-lived workflows into transactions with compensating actions, but compensation does not logically erase every real-world consequence [6]. Gray and Lamport show how coordinated commit depends on agreement across participants [7]. In practice, idempotent APIs use stable request identities to make retries less dangerous [8]. The IBCC contribution is not a new consensus algorithm or an exact-once guarantee for arbitrary networks. It combines endpoint-level idempotency with action-intent, state-version, policy-version, and expenditure bindings inside one enforceable commit precondition.

Security principles and positioning

Classic protection principles place enforcement in a trusted reference mechanism rather than in the discretion of an untrusted program [11]. Proof-carrying code similarly motivates independently checkable evidence [12], although IBCC certificates are not proof-carrying code and do not certify the semantic correctness of arbitrary programs. The NIST AI Risk Management Framework addresses broader governance, organizational and risk processes [13]; IBCC supplies a lower-level control contract compatible with, but not sufficient to implement, those practices. Table 1 specifies the intended novelty boundary. Tan [14] examines control decay, Tan [15] studies verification debt, and Tan [16] proposes a governance-aware engineering architecture. These related contributions inform the motivation but do not independently verify the transaction-level guarantees established here.

Table 1. Relationship to established research and the present contribution.

Foundation	Established contribution	Specific extension proposed here
LLM agents [1–5]	Tool calling, interactive evaluation, injection threat models	Execution-time effect integrity independent of model compliance
Sagas and commit [6,7]	Compensation and distributed agreement	Per-effect certificate tied to intent and observed resource version



Idempotent APIs [8]	Retry-safe request identity	Bind stable key to the principal, intent, and action digest
Protection/PCC [11,12]	Reference monitoring and checkable artifacts	Typed commit authorization that is enforceable at a backend adapter
TAN (2026) [10]	Runtime autonomy-risk scoring	State-dependent transaction invariants rather than risk categorization

METHODOLOGY AND FORMAL DESIGN

Design-science approach and threat model

The study uses constructive formalization followed by deterministic fault injection and bounded schedule exploration. A workflow is a directed acyclic graph of potential side-effecting operations. The agent proposes an operation, but only a trusted authority may issue a certificate, and only a trusted commit adapter may apply an effect. The adversary can influence model output and untrusted tool content, replay or reorder requests, and trigger lost acknowledgments; it cannot forge a correctly implemented signature or bypass the adapter. This is a deliberately explicit boundary: a compromised signer, bypassed adapter, or broken storage isolation invalidates the main guarantee.

	$G = (V, E), \quad a_i = (p_i, r_i, o_i, x_i, I_i, k_i, v_i).$	(1)
--	--	-----

Here p denotes the authenticated principal, r the target resource, o the operation type, x its cost or exposure quantity, I the independently authorized intent identifier, k the stable idempotency key, and v the observed resource version. A workflow may contain several such actions, but global atomicity across all vertices is not assumed. An effect is a state-changing operation; merely returning a model output is not an effect.

State and certificate schema

The trusted adapter's state is represented by resource versions v , remaining budgets b , the durable idempotency map D , the append-only evidence ledger L , and policy version π . Resource-version validation applies to precisely the material fields on which the authorization decision depends; a coarse global version may reject unnecessarily, whereas an incomplete read set may miss material changes.

	$S_t = (v_t, b_t, \mathcal{D}_t, \mathcal{L}_t, \pi_t).$	(2)
--	--	-----

For each action, the authority signs a certificate c carrying canonical hashes of the approved intent and normalized action, the principal and resource bindings, resource version, maximum permitted amount, expiry, idempotency key, policy version, and signature. Domain separation and canonical serialization are required to prevent two semantically different requests from sharing a misleading digest. The certificate is not self-authorizing: only the trusted authority can produce a valid signature.

	$c_i = (H(I_i), H(a_i), p_i, r_i, v_i, b_i^{\max}, t_i^{\exp}, k_i, \pi_i, \sigma_i).$	(3)
--	--	-----

Cryptographic collision resistance and unforgeability are assumptions, not measured properties. A certificate cannot elevate an untrusted text instruction to real user consent. The trusted issuance process must verify the principal's mandate, spending scope, and policy independently. An action that modifies more than one resource must declare an appropriate read/write set and use a transactional adapter encompassing the guarded invariants.

Table 2. Normative certificate field definitions and validation obligations. Field names are illustrative; the signed representation must specify versioned canonical encoding.

Field	Type / requirement	Check at commit boundary
schema, audience	Versioned integer; service identifier	Reject unsupported schema or destination (domain separation)
principal, intent_hash	Authenticated principal; 256-bit digest	Verify caller scope and pre-authorized user mandate
action_hash, resource	Canonical SHA-256 digest; resource ID	Recompute bound action and reject substitution
resource_version / read set	Version(s) of material state read	Compare with committed state under transaction lock
policy_version	Trusted authorization epoch	Reject new commits after invalidation or reissue
max_amount, expires_at	Nonnegative amount; trusted expiry	Enforce live aggregate budget and expiry at commit
idem_key	High-entropy stable operation identifier	Uniqueness and principal/digest collision detection
signature	Issuer key ID plus authenticated signature	Check signature over domain-separated canonical payload

Certificate construction and trust chain. A production certificate format must specify a unique message schema, canonical byte representation, a cryptographic domain separator, algorithm and issuer key identifiers, and a protected user-consent reference. The action digest must cover all material side-effect parameters, not merely a natural-language description. An issuer first checks authenticated consent, spending rights, and current policies. An adapter then independently checks the signature and certificate scope before applying any new effect. In the executable local example the digest is SHA-256 and the demonstration signature is HMAC-SHA-256 with a fixed test-only secret. This is a testing convenience, not a secure deployed key-management design.

Guarded commit and replay semantics

The adapter validates signature, canonical intent/action binding, the current policy, version freshness, amount bounds, and the live budget. It also compares certificate policy version with the current version or explicitly enforces a stricter current policy. In Equation (4), Sig verifies the signed certificate, Bind checks its canonical intent/action digest, and Auth enforces the current policy and principal mandate, including certificate-policy compatibility. Fresh requires the certified version to equal the live material-resource version. Bound requires the amount to be no greater than both the certificate limit and the live available budget, while Live requires the certificate not to have expired. These predicates are evaluated at the same linearization point as the modification.

	$\text{Valid}(a_i, c_i, S_t) = \text{Sig}_i \wedge \text{Bind}_i \wedge \text{Auth}_i \wedge \text{Fresh}_i \wedge \text{Bound}_i \wedge \text{Live}_i.$	(4)
--	--	-----

A duplicate key must be resolved against the durable ledger before considering a new effect. A repeated request with the same key and identical canonical action receives the prior receipt. A repeated key with a different principal or action digest is rejected as a key collision. Equation (5) explicitly gives the replay and digest-collision branches priority over new-key validation. Receipts must not disclose information to an unauthorized requester; retrieval of an existing receipt requires authenticated scope checking.

	$\text{Commit}(S_t, a_i, c_i) = \begin{cases} (S_t, \text{receipt}), & k_i \in \mathcal{D}_t, \text{ Match}; \\ (S_t, \text{reject}), & k_i \in \mathcal{D}_t, \neg \text{Match}; \\ (S_t, \text{reject}), & k_i \notin \mathcal{D}_t, \neg \text{Valid}; \\ (S_{t+1}, \text{receipt}), & k_i \notin \mathcal{D}_t, \text{ Valid}. \end{cases}$	(5)
--	---	-----

On a successful new-key commit, the adapter atomically decreases the relevant budget, increments the corresponding version, stores the idempotency key and digest, and records the receipt. Across real services, this atomicity requires an authoritative database transaction or an endpoint exposing equivalent conditional-write and idempotency semantics. Recording a key only in the agent process is insufficient.

	$b_{t+1}(r_i) = b_t(r_i) - x_i \geq 0, \quad v_{t+1}(r_i) = v_t(r_i) + 1.$	(6)
--	--	-----

Trust boundary: the language model never possesses direct commit authority

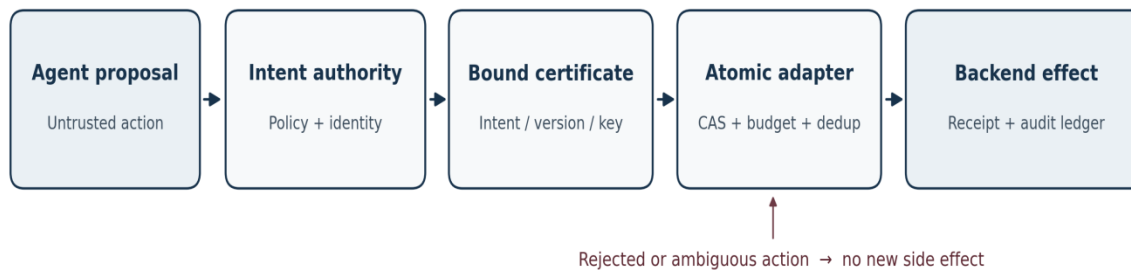


Figure 1. IBCC trust boundary. The AI proposes actions but cannot directly commit side effects. The adapter executes only certificate-bound, atomic state transitions.

Failure handling and uncertain outcomes

A lost acknowledgment is not evidence that the operation failed. On timeout, the client repeats the same canonical action with the same key and queries the authoritative receipt. If the backend cannot reconcile the key to its commit status, the safe behavior for high-consequence, non-idempotent effects is to hold and escalate, not to issue a fresh key. Irreversible downstream actions may warrant a human resolution pathway; compensating actions in a saga remain separate transactions and do not constitute exact reversal [6].

	$\text{Release}(a_i) = \begin{cases} \text{commit,} & \text{valid and atomically enforceable;} \\ \text{hold,} & \text{uncertain and not reconcilable;} \\ \text{reject,} & \text{invalid authority or state.} \end{cases}$	(7)
--	---	-----

Algorithm 1: trusted adapter procedure

Input: authenticated principal p, canonical action a, signed certificate c, durable state S.

1. Verify caller scope; canonicalize a and recompute the action and intent digests.
2. Begin a serializable/linearizable transaction (or obtain equivalent exclusive conditional-write access). Inside that boundary, read k from the authoritative deduplication map. If k exists, authenticate receipt access and return the previous receipt only when principal and canonical digest match; reject any key collision.
3. For a previously unseen key, verify certificate signature, schema, audience, expiry, principal and intent binding, live policy epoch, declared read-set versions, and canonical action digest within the same protected transaction.



4. Check amount and aggregate live budget. Reserve the key using an atomic uniqueness constraint or compare-and-insert, ensuring two concurrent requests cannot both consider it new.
5. Apply a single locally atomic backend effect; persist the budget decrement, resource versions, deduplication key, digest, receipt, and audit row before committing together. On any fault, roll back all protected mutations.
6. Commit the transaction and return the receipt; on a conflict abort, refresh state and require a newly authorized certificate.
7. On network uncertainty, reconcile by k. Never substitute a new key solely because the acknowledgment is missing.

Commit-state ordering and implementation preconditions

For new-key execution, the atomic linearization point is the successful commit of a transaction that contains (i) authorized certificate validation against current data, (ii) uniqueness-enforced key insertion, (iii) a budget-preserving resource effect, and (iv) the authoritative receipt. An implementation must not place a deduplication lookup outside the protected boundary and then rely on that lookup for uniqueness: two simultaneous requests could both observe an absent key. The key database constraint and the resource/budget transaction must share a failure domain, or equivalent endpoint semantics must be established. For duplicate keys, receipt retrieval is a non-effecting read and requires authenticated principal/action binding even if the certificate has subsequently expired or the policy epoch has advanced. A conflicting binding is always rejected.

On a state mismatch, the adapter aborts, reports a conflict and requires a fresh certificate issued against the new state. On an uncertain network outcome, the requester first consults the durable key-to-receipt record. The transaction guarantees do not survive attempts to perform a nontransactional remote effect inside a local SQL transaction; those require endpoint-provided idempotency or a separately analyzed outbox/reconciliation design. An append-only audit entry alone does not establish a valid user mandate or a tamper-proof log.

FORMAL SAFETY RESULTS

Theorem 1: conditional commit invariants

Theorem 1 (single-authority effect integrity). Assume (A1) all relevant side effects pass through the trusted adapter; (A2) certificate signatures and canonical digests cannot be forged or ambiguously encoded; (A3) authorization and version predicates are evaluated at the atomic effect linearization point; (A4) budget decrement, key insertion, resource update, and receipt persistence are atomic and durable; and (A5) receipt replay is authorized and keyed to the same action digest. Then every committed effect is authenticated, authorized by the enforced policy, fresh at its linearization point, within its protected budget, and at most once for each key/action pair. The key must be unique to a genuine approved operation: a trusted issuer that assigns multiple unrelated keys to the same semantic intent can still authorize repeated real-world effects, which is outside the theorem.

	$\forall e \in \mathcal{E}_{\text{commit}}: \text{Authorized}(e) \wedge \text{Fresh}(e) \wedge \text{WithinBudget}(e) \wedge \text{Unique}(e).$	(8)
--	---	-----

Proof. Initially no effect exists and the protected budget is nonnegative. Suppose the property holds through step t . A rejection or matching receipt replay makes no state change, so the invariant persists. For a new commit, A2 and A3 establish authorization, action binding and freshness at the same instant; A4 deducts only an amount not exceeding the remaining budget, preserving nonnegativity. A4 also makes key insertion indivisible from the effect. Any later request with the same key encounters the stored digest and cannot reapply the effect, while an inconsistent digest is rejected. Thus the inductive property holds at $t + 1$. By induction, all adapter-mediated commits satisfy Equation (8). QED.



The theorem addresses per-effect safety, not global workflow correctness. A certificate may authorize an action that is strategically unwise; the model's factual assumptions may be incorrect; and two individually authorized actions may have an undesirable combined consequence if a shared constraint was omitted. Security therefore depends on complete invariant specification and trusted issuance.

Proposition 1: blind-retry indistinguishability

Proposition 1. Consider an asynchronous requester and a non-idempotent service with no durable request-key lookup, no observable commit status, and potentially lost acknowledgments. After a timeout, the requester cannot simultaneously guarantee eventual execution when the first attempt did not commit and guarantee no duplicate effect when the first attempt did commit.

Proof. Construct two executions with identical observable requester histories. In execution A, the request is lost before the service applies the effect. In execution B, the service commits the effect but its acknowledgment is lost. Both yield the same timeout observation. A policy that never retries fails eventual execution in A; a policy that retries may duplicate the non-idempotent effect in B. Because the requester cannot distinguish A from B with the available observations, no decision rule satisfies both guarantees in these executions. QED. The result is deliberately restricted to this failure model; endpoint idempotency or status reconciliation changes the premises.

Boundaries of compositional safety

When a workflow spans multiple independent services, each service can satisfy Theorem 1 locally while the overall workflow remains partially committed. Cross-service atomicity additionally requires a coordinating transaction protocol or domain-specific compensation, and some real-world effects are irreversible [6,7]. Moreover, version-based freshness is limited to the declared read set. A certificate signed before a policy change must be invalidated or revalidated under the new policy. The protocol accordingly distinguishes at-most-once effect application from successful end-to-end completion.

Proposition 2: concurrent uniqueness through atomic deduplication

Proposition 2. Suppose all requests for one idempotency key access a durable uniqueness-enforced map within a serializable transaction that atomically persists the resulting local effect and its receipt. Regardless of their arrival order, at most one transaction can create a new effect for that key; all later matching requests may only retrieve its prior receipt.

Argument. Assume two distinct committed transactions each create an effect for the same key. Each would also have committed insertion of that key into a map with a unique-key constraint. Transactional serializability orders the two commits, so the later transaction must observe the existing binding or fail its uniqueness insertion. It cannot commit a second effect. This contradicts the assumption. The argument depends on a common authoritative transactional state, not on application-process locks, and does not establish global exactly-once delivery for nontransactional external systems.

REPRODUCIBLE COMPUTATIONAL EVALUATION

Synthetic experiment and policies

A standalone Python 3 standard-library simulator generated 50,000 independent proposed side-effect episodes using the fixed pseudorandom seed 20261010. For each episode, the model samples five binary fault flags independently: unauthorized request (9%), stale context (11%), overbudget request (7%), duplicate request (12%), and lost acknowledgment (8%). Duplicate requests and lost acknowledgments each produce one extra attempt for an accepted request. These parameter values are illustrative design choices, not estimated field frequencies. The unit of analysis is a proposed effect episode rather than an entire multi-agent workflow or a real LLM response.

Five configurations were compared on precisely the same sampled episodes: Direct tool call (no gate, no freshness, no deduplication); Policy gate (authorization and amount rules); Gate plus idempotency; Freshness plus idempotency (without authorization or amount gate); and the full IBCC combination. For tractability, the simulator treats checks as reliable and creates a new effect for each extra attempt in configurations without deduplication. IBCC assumes successful atomic enforcement as required by Theorem 1. This is a component-ablation test of the protocol's logic, not an independent benchmark against deployed commercial systems.

An episode is eligible if it is authorized, fresh, and within budget. An unsafe episode is one that commits any unauthorized, stale, overbudget, or duplicated effect. Clean valid completion means an eligible episode creates exactly one effect. Thus a configuration cannot be declared useful merely because it rejects everything. Coverage is normalized against the same set of eligible opportunities. In Equations (9) and (10), the numerators count, respectively, eligible episodes completed cleanly and episodes with at least one unsafe effect; the denominators count all eligible opportunities and all simulated episodes. The simulator models the safeguards as exact Boolean enforcement decisions, so the full-model zero-failure count is a designed property of its transition rules. It should not be read as an externally estimated probability of attack resistance.

	$\text{Coverage} = \frac{N_{\text{clean}}}{N_{\text{eligible}}}.$	(9)
--	---	-----

	$\text{UER} = \frac{N_{\text{unsafe}}}{N_{\text{total}}}.$	(10)
--	--	------

With independent sampled eligibility faults, the expected eligible fraction and expected extra attempts for an unguarded accepted episode are given by Equations (11) and (12). For the gate-plus-idempotency ablation, stale context is its only remaining unsafe class, yielding Equation (13). These analytical quantities provide internal checks against the simulator, rather than independent validation.

	$\text{Pr}(\text{eligible}) = (1 - p_u)(1 - p_s)(1 - p_b).$	(11)
--	---	------

	$\mathbb{E}[N_{\text{extra}} \mid \text{unguarded accepted request}] = p_d + p_\ell.$	(12)
--	---	------

	$\text{Pr}(\text{unsafe episode; gate + dedup}) = (1 - p_u)(1 - p_b)p_s.$	(13)
--	---	------

Episode-rate uncertainty is summarized descriptively using the Wilson score interval (Equation 14), where $\hat{p} = k/n$, k is the count of observed unsafe episodes, n is the number of sampled episodes, and $z = 1.96$. This interval characterizes finite simulation sampling under the stated generator; it does not capture uncertainty about whether the fault model reflects the real world. In particular, observing zero simulated failures is not proof of zero failures outside the assumed enforcement model.

	$\text{CI}_{0.95} = \frac{\hat{p} + \frac{z^2}{2n} \pm z \sqrt{\frac{\hat{p}(1 - \hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}.$	(14)
--	---	------

Table 3. Enforcement ablations and fault classes excluded by each configuration. A Yes entry denotes a modeled control, not demonstrated robustness in production.

Configuration	Authorization and amount	Version freshness	Durable replay suppression
Direct tool call	No	No	No
Policy gate	Yes	No	No
Gate + idempotency	Yes	No	Yes
Freshness + idempotency	No	Yes	Yes
IBCC full	Yes	Yes	Yes, inside atomic commit

Results across 50,000 episodes

Table 4. Deterministic synthetic testbench results. All rates use the 50,000 episode denominator except clean coverage.

Protocol	Admitted	Unsafe episodes	Unsafe rate	Clean valid coverage
Direct tool call	50,000	19,319	38.64%	81.26%
Policy gate	42,418	11,737	23.47%	81.26%
Gate + idempotency	42,418	4,660	9.32%	100.00%
Freshness + idempotency	44,513	6,755	13.51%	100.00%
IBCC (full)	37,758	0	0.00%	100.00%

Of 50,000 proposed episodes, 37,758 were eligible under the sampled generator. The direct tool-call configuration committed an unsafe effect in 19,319 episodes (38.64%). The policy gate alone reduced this to 11,737 (23.47%), but could not prevent stale effects or duplicate commits. Combining the gate with idempotency reduced unsafe episodes to 4,660 (9.32%), exactly the residual stale-context class anticipated by Equation (13). A freshness-plus-idempotency configuration, which does not enforce principal or budget limits, left 6,755 unsafe episodes (13.51%).

Under the enforced assumptions of the synthetic model, IBCC accepted 37,758 episodes, committed exactly one effect for each, and recorded zero unsafe episodes. Its clean coverage was 100.00% of the 37,758 eligible opportunities, not 100% of all proposed episodes. The zero-event Wilson 95% upper limit is 0.008% for this artificial generator; the formal safety result instead rests on Theorem 1 and its prerequisites. No real latency, token cost, commercial model accuracy, or human approval effort was measured.

Stale-context sensitivity

A separate fixed-seed sensitivity sweep used 20,000 episodes per condition and varied the stale-context injection probability from 0% to 30%, leaving the other fault rates unchanged. Figure 2 plots unsafe-episode frequency against staleness prevalence. The gate-plus-idempotency failure frequency increases approximately linearly with staleness, while the full protocol remains at zero by construction. The exercise identifies a missing enforcement dimension; it is not evidence that future real-world adversaries or service implementations are captured by the simulator.

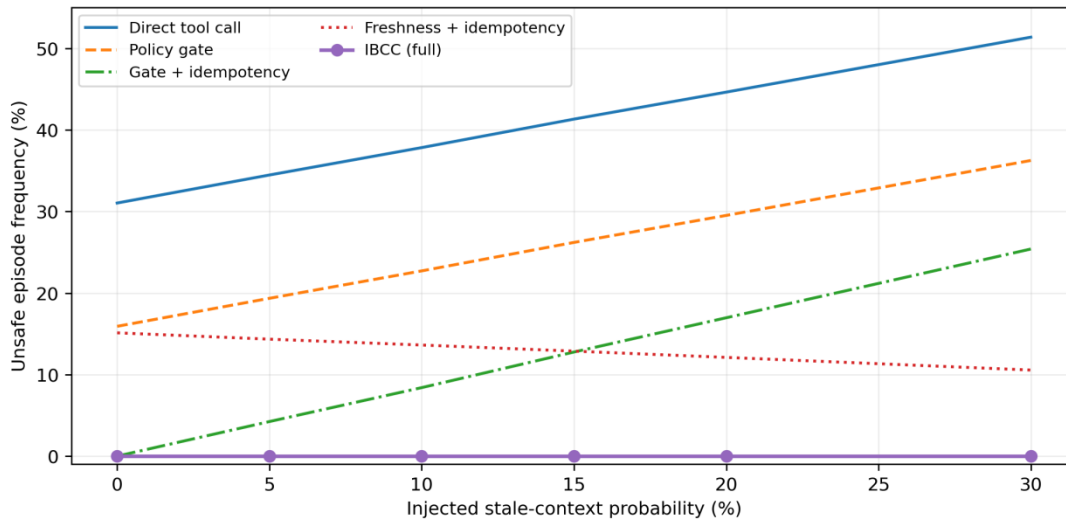


Figure 2. Sensitivity of unsafe-episode frequency to the injected stale-context probability under five synthetic enforcement configurations.

Exhaustive two-client budget interleavings

A second test enumerated all legal interleavings of two clients, each performing a read followed by a commit, against a shared initial budget of 100 units. Each client requests 70 units. In the split check/act baseline, a successful read authorizes a later ungarded debit; in the IBCC adapter, the debit also requires an atomic contemporaneous budget check. There are six order-preserving interleavings in total.

$$N_{\text{interleavings}} = \binom{4}{2} = \frac{4!}{2! 2!} = 6. \quad (15)$$

Table 5. Exhaustive two-client schedule exploration with a shared 100-unit budget.

Enforcement mechanism	Schedules explored	Schedules exceeding budget	Maximum final deficit
Split check/act	6	4	40 units
IBCC atomic conditional	6	0	0 units

In four of the six legal orderings, both clients read sufficient budget before either commits and the split check/act design reaches a negative balance of 40 units. The atomic conditional operation prevents all such overruns by rechecking remaining budget at commit. This is bounded schedule exploration of a deliberately small concurrency model, not a proof for arbitrary distributed topologies. The general budget property depends on A3 and A4.

Independent SQLite transaction-adapter integration tests

To test the commit protocol against actual transactional storage rather than only the synthetic Boolean simulator, a compact Python 3 sqlite3 reference adapter was built. Each call opens a fresh connection, executes BEGIN IMMEDIATE, authenticates the certificate, resolves a key or validates new-commit conditions, updates the resource and shared budget, inserts a uniquely keyed receipt, and commits or rolls back in one SQLite transaction. WAL mode and a busy timeout support threaded contention. The experiment uses an explicitly non-production HMAC test key, structured JSON canonicalization, and a deterministic injected clock. All effects are local database mutations, not payments or remote API calls. The complete implementation, test harness, and captured results accompany the reproducibility package.

Table 6. Actual integration-test outcomes from the independent SQLite adapter (12 tests; all passed).

Test group	Injected scenario / outcome	Observed result
T1: duplicate-key race	8 simultaneous requests with identical certificate and key	1 COMMIT; 7 REPLAY; 1 effect
T2: shared-budget race	8 distinct resource/key requests, budget of 3 units	3 COMMIT; 5 REJECT_BUDGET; 0 overspend
T3-T5: identity binding	Action substitution; modified signed certificate; wrong principal	3 rejected, no effect
T6-T7: temporal validity	Expired certificate; changed policy epoch	2 rejected, no effect
T8-T10: state and replay	First valid commit; stale second key; conflicting reuse	1 commit; 2 rejections
T11: uncertain response	Resubmit same key after simulated lost reply and policy change	Prior receipt replayed; no second effect
T12: transactional fault	Raise exception after SQL writes, before commit	0 effects; 0 receipts; budget unchanged

The executed harness reports 12 of 12 passing assertions. In T1, an eight-worker barrier produces one committed effect, seven receipt replays, and a budget reduction from 10 to 9. In T2, eight independent resource actions contend over a three-unit shared budget: exactly three effects commit, five are rejected, and the final balance is zero. T12 confirms that an exception between provisional SQL writes and COMMIT rolls back effect, key, and budget mutations. These observed results close the particular race condition highlighted in Algorithm 1, but they are not a statistical failure-rate study or exhaustive schedule proof. SQLite serializes writers; no latency, throughput, multiprocess partition, adversarial cryptographic strength, or multi-service consistency claims are drawn.

Reproducibility and auditability

The supplied Python 3 package reproduces the original 50,000 synthetic episodes, six two-client schedules, and 12 SQLite integration tests. Fixed seeds, fault parameters, five baseline definitions, all counts, and JSON/CSV outputs are documented in the accompanying README and Appendix B. No production service, LLM inference, or human participant was involved.

DISCUSSION

Why authorization and execution must be separate

A model's decision to call a tool is a request for action, not evidence of a mandate. Identity binding prevents a tool call from silently inheriting arbitrary permissions. Intent binding prevents an authentic principal's broad access from being repurposed for an unrelated action. Freshness prevents an authorized but outdated plan from being committed after a material state change. Idempotency prevents retried delivery from being mistaken for independent intentions. Budget atomicity prevents concurrent callers from exhausting the same allowance twice. These properties are logically separable, as the ablation results illustrate.

The mechanism is related to reference-monitor thinking [11] and to the value of checkable execution evidence [12], but it does not assert that signed certificates make agents generally truthful or safe. They constrain effects to verifiable authority and state conditions. Risk management, model evaluation, human supervision, incident response, and security of the certificate issuer remain necessary [10,13].



Engineering deployment matrix

Table 7. Minimum endpoint capabilities and prudent action under commit uncertainty.

Endpoint class	Available guarantee	Recommended execution policy
Atomic conditional write + durable key	Per-resource freshness, budget and at-most-once under assumptions	Automated commit with IBCC and authenticated receipt
Durable key and status query, no shared transaction	At-most-once for one endpoint; no global multi-service atomicity	Commit each leg, log dependency, reconcile; use saga if appropriate
No idempotency but authoritative status lookup	Conditional reconciliation, dependent on complete observability	Single attempt, query status on timeout, escalate uncertain cases
Non-idempotent, non-reconcilable external action	Cannot guarantee both progress and nonduplication after lost reply	Human-reviewed single attempt; hold after uncertainty

A secure implementation should additionally specify certificate expiry and clock-skew handling, key retention windows at least as long as the permitted retry horizon, principal-specific receipt access, policy invalidation, cryptographic key rotation, canonical action versioning, evidence retention, rollback behavior, and denial-of-service rate limits. Hash equality alone cannot establish semantic equivalence if canonicalization is unstable. An audit log also must not be treated as proof of immutable storage without tamper-evident controls.

Relevance and limitations

IBCC is applicable to payment approval, infrastructure provisioning, customer-record modification, purchasing, automated service tickets, and other cases where AI agents cross an externally meaningful state boundary. For low-stakes read-only calls, its additional infrastructure may not be proportionate. Decision rules for allocating oversight should consider impact, reversibility, endpoint capability, and the costs of false rejection rather than imposing the same barrier on every call. Unlike the governance-decay analysis of Tan [14] or the verification-debt perspective of Tan [15], these results are narrowly about the enforceability of a transaction boundary, not the durability of enterprise-wide human oversight. Tan [16] supplies a broader enterprise engineering context; it does not independently validate the present SQLite implementation.

Six limitations remain consequential. First, synthetic probabilities are illustrative and do not estimate field frequencies. Second, the original full-protocol simulator implements ideal checks by construction; zero unsafe episodes is therefore an internal consistency result rather than independent empirical evidence. Third, the added SQLite integration tests use an authoritative local database, a test-only HMAC key, fixed scenarios, and writer serialization; they do not validate physical payment effects, cryptographic key governance, cross-region partitions, or throughput. Fourth, only the declared read set can be checked for freshness, and omitted shared constraints remain vulnerabilities. Fifth, signed consent bindings cannot independently establish the semantic correctness of an AI plan or rule out downstream social harms. Sixth, per-endpoint at-most-once effects do not imply globally atomic multi-service completion. Future validation should use authenticated production-grade signatures, mechanized model checking, realistic failure schedules, stateful cloud endpoints, service-level evidence, and measured throughput and rollback behavior.

CONCLUSION

Safe tool-using AI requires more than detecting problematic model outputs. Before an autonomous side effect occurs, a trusted system must establish whose authority is being exercised, which intent and action were approved, which resource state was observed, how much may be consumed, and whether the same intention has already committed. Intent-bound commit certificates combine these fields with atomic adapter enforcement. Under explicitly stated assumptions, the resulting protocol establishes per-effect authorization,



freshness, protected-budget integrity, and at-most-once execution for a bound idempotency key. A separate indistinguishability argument shows why blind retries cannot provide both guaranteed completion and nonduplication for opaque non-idempotent services. In a fully disclosed synthetic evaluation, the protocol eliminates unsafe effects without rejecting eligible opportunities, while ablations reveal failures from missing controls. These results motivate production-level validation rather than claims of universal deployment readiness. The research direction is to make autonomous agency governable at the exact boundary where a proposed action becomes an irreversible external fact. The supplementary SQLite study also demonstrates that its local implementation resists the tested key races, budget races, and mid-transaction rollback failures. Formal assumptions, synthetic results, and observed local integration behavior are deliberately distinguished; broader distributed guarantees remain outside the study.

DECLARATIONS

Funding: No external funding was used for the preparation of this study.

Competing interests: The author declares no competing interests.

Ethics: The study uses only artificial scenarios and a deterministic software testbench. It involved no human participants or identifiable personal data, and no human-subject research approval was required for the reported work.

Data and code availability: The two executable Python experiments, original numerical outputs, independent SQLite adapter and integration tests, and machine-readable test results are supplied as an accompanying reproducibility archive. Only synthetic data and local transactional effects were used; no production system or external dataset was evaluated.

Author contribution: Kwan Hong TAN is the sole named author and takes responsibility for reviewing the study design, formal claims, numerical results, and submitted manuscript.

REFERENCES

1. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. International Conference on Learning Representations. <https://arxiv.org/abs/2210.03629>
2. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems, 36. <https://arxiv.org/abs/2302.04761>
3. Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., et al. (2024). AgentBench: Evaluating LLMs as agents. International Conference on Learning Representations. https://proceedings.iclr.cc/paper_files/paper/2024/hash/e9df36b21ff4ee211a8b71ee8b7e9f57-Abstract-Conference.html
4. Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., & Tramèr, F. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. Advances in Neural Information Processing Systems, 37. <https://doi.org/10.52202/079017-2636>
5. Zhan, Q., Liang, Z., Ying, Z., & Kang, D. (2024). InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. Findings of the Association for Computational Linguistics: ACL 2024, 10471–10506. <https://doi.org/10.18653/v1/2024.findings-acl.624>
6. Garcia-Molina, H., & Salem, K. (1987). Sagas. ACM SIGMOD Record, 16(3), 249–259. <https://doi.org/10.1145/38714.38742>
7. Gray, J., & Lampert, L. (2006). Consensus on transaction commit. ACM Transactions on Database Systems, 31(1), 133–160. <https://doi.org/10.1145/1132863.1132867>
8. Featonby, M. (2021). Making retries safe with idempotent APIs. Amazon Builders' Library. <https://aws.amazon.com/builders-library/making-retries-safe-with-idempotent-APIs/>



9. Abadi, D. J. (2012). Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *Computer*, 45(2), 37–42. <https://doi.org/10.1109/MC.2012.33>
10. TAN, K. H. (2026). Runtime assurance for enterprise agentic AI systems: A policy-gated control model with quantitative autonomy-risk scoring. *World Journal of Advanced Research and Reviews*, 31(1), 512–522. <https://doi.org/10.30574/wjarr.2026.31.1.1872>
11. Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278–1308. <https://doi.org/10.1109/PROC.1975.9939>
12. Necula, G. C. (1997). Proof-carrying code. *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 106–119. <https://doi.org/10.1145/263699.263712>
13. Tabassi, E. (2023). Artificial intelligence risk management framework (AI RMF 1.0). NIST AI 100-1. National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.AI.100-1>
14. Tan, K. H. (2026). The governance half-life of agentic artificial intelligence controls: A dynamic assurance framework for control decay, residual risk, and revalidation. *International Journal of Web of Multidisciplinary Studies*, 3(7), 217–229. <https://doi.org/10.71366/ijwos03072654084>
15. Tan, K. H. (2026). The coordination compression trap: A dynamic model of AI-enabled productivity, verification debt, and organisational resilience in global knowledge firms. *Journal of Global Economics, Management and Business Research*, 18(3), 256–278. <https://doi.org/10.56557/jgembr/2026/v18i310956>
16. Tan, K. H. (2026). Governance-aware agentic AI for enterprise engineering systems: A design-science reference architecture and quantitative risk-control model. *International Journal of Science Engineering and Technology*, 14(3). <https://doi.org/10.5281/zenodo.21264295>

Appendix A. Experimental Protocol (for transparency)

Experiment identifiers. Main simulation seed: 20261010; main episodes: 50,000; sensitivity seed: 20261081; sensitivity episodes: 20,000 per rate. Fault parameters: $p(\text{unauthorized}) = 0.09$; $p(\text{stale}) = 0.11$; $p(\text{overbudget}) = 0.07$; $p(\text{duplicate}) = 0.12$; $p(\text{lost acknowledgment}) = 0.08$. Staleness is replaced by 0, 0.05, 0.10, 0.15, 0.20, and 0.30 in the sweep. Random flags are sampled independently for each episode. All acceptance and unsafe outcome logic is directly inspectable in the supplied script.

Counting rules. Accepted episodes are proposals for which the protocol permits a first side effect. In the absence of deduplication, each injected duplicate or lost acknowledgment adds one committed effect; with deduplication each accepted episode has one effect. A side effect is unsafe when its originating episode fails authorization, freshness, or budget rules, or when it represents a redundant replay. Clean eligible coverage is calculated only among authorized, fresh, within-budget proposals. Zero unsafe events under the full protocol is therefore a property of the modeled enforcement semantics, not a measured failure probability for real services.

Appendix B. Independent Transaction-Adapter Reproduction

Reproduction environment. Python 3.10 or newer with the standard library and local SQLite support. From the supplementary archive, run "python ibcc_experiment.py" for the original 50,000-episode experiment, and run "python ibcc_sqlite_tests.py" in the integration subdirectory for the second study. The tests generate temporary WAL-mode databases, assert the exact outcomes listed in Table 6, then clean up. The captured JSON output is provided for comparison.

Primary transactional specification. The receipts table defines key k as PRIMARY KEY; resources tracks a version and effect count; budgets enforces nonnegative balance; audit rows have a unique key. Every new operation starts with BEGIN IMMEDIATE and ends in COMMIT only after all guarded mutations have been persisted within the same transaction. On certificate failure, conflict, or injected exception, ROLLBACK removes all uncommitted updates. Receipt lookup for duplicate keys occurs while the transaction holds the corresponding serialization protection.



Boundary note. The local reference adapter models commit integrity for SQLite-controlled state. A call to an external payment processor from inside the SQLite transaction would NOT inherit these atomic semantics. Test HMAC material is intentionally fixed and public within the source for reproducibility, so it must not be reused for authenticating real users or systems.