The Sealed Harbor

Mutually Confidential Computation with Explicit, Gated, Bounded Releases

Paper 4 of the Harbor program — the four-pillar assurance argument, executed

Prepared for Erich Owens

August 2026

Abstract

Two parties who will share neither data nor model can nonetheless obtain an attributable, policy-bound joint computation, with every information release explicit, gated, and bounded — and, on the stated empirical hypothesis of §1 that token-level taint through a generative model is not soundly definable, the security boundary is the declassification gate rather than token-level taint. We present the clean-room design for tool-using LLM agents (dual-attested key release, whole-worker taint, two declassification gates) and its assurance argument as four independently verified pillars: (1) observational noninterference modulo declassification, verified exhaustively on the finite clean-room model with a mutation suite that catches both canonical breaks; (2) the supervisory-control result that fixes *which* boundary can be enforced at all — the channel, never the token; (3) ε-conservation of the release ledger, with differential-privacy composition giving the conserved sum its meaning; and (4) a canary detection theory with a quotable operating curve and a sequential-test alarm clock. We price the residual honestly: an information-theoretic leakage budget of $q \cdot b$ bits across q jobs through any b-bit release channel, before timing channels, which are out of model. *Express lane: the one-breath claim of each pillar opens its section; the formal statement is in the box below it; a reader in a hurry needs nothing else.*

1 The problem: Derek and Erin

The scene. Derek owns sensitive financial data D. Erin owns a valuable agent system M — weights, prompts, skills, tools, orchestration. Both want the answer $y = f_M(D)$. Neither will hand over the crown jewels: Derek must not receive M, Erin must not receive D, and the cloud operator hosting the computation must receive neither. Every week some enterprise negotiates this exact standoff with lawyers and air-gapped laptops; the question is whether a *computational* contract can replace the lawyers' one — and what, precisely, it can promise. The systems literature has answered a narrower version of this question already: Ryoan [24] confines a proprietary module inside an attested enclave so that a data owner and a code owner who distrust each other both keep their secrets, and we claim none of that as new. What Ryoan buys its confinement with is a module that sees its input once and keeps no state — a price a looping, tool-calling agent cannot pay, and the whole reason this paper needs a budget and a detector where Ryoan needed neither. Section 10 is explicit about the line, row by row.

The tempting pitch — and the one an early product review of this program actually wrote — is a "silicon-enforced NDA": tag Erin's agent `[TAINT:HIGH]` the moment it reads Derek's data, drop any tainted egress packet, and declare that the data "mathematically cannot phone home." This paper's first job is to explain why that sentence must never be shipped, and its second job is to prove the sentence that replaces it. *[Empirical hypothesis; no formal statement is offered here.]* Token-level taint through an LLM is not soundly definable: the model taints everything it writes with everything it read, so a taint tracker faces a binary choice between marking every output token (useless) and missing semantic flows (unsound). This is a design rationale, not a

theorem, and it is the one critical claim in this paper that carries no proof and no script. It rests on an empirical hypothesis about generative models — we know of no sound token-granularity flow analysis for one, and the burden of exhibiting one falls on any design that assumes it — standing on the well-known static-analysis result underneath: certifying implicit flows soundly forces exactly this dichotomy, the label creep of the Denning certification lattice [12] and the extra restrictiveness a sound purely dynamic monitor must accept relative to a static one [13]. Every later use of the premise (§4, §10) inherits it by reference from here; none re-derives it. A secret can ride out in wording, punctuation, error messages, output length, or timing; no regex, PII scanner, or second LLM can certify that arbitrary prose contains no secret. And an ordinary remote model API is *itself* a trusted reader of plaintext — cryptography cannot repair sending Derek's data to a third party's inference endpoint.

The honest architecture therefore controls the *channel*, not the token, and §2 lays it out. The design converts the question a CISO will actually ask — *how do I know the wall has no other holes?* — into four provable claims, one per section below: the gate is the only hole (§3); the gate sits on the only boundary that can be enforced at all (§4); what passes through it is metered by a conserved budget (§5); and what tries to sneak around the meter is caught with quotable probability and latency (§6). Section 7 composes the pillars into a single leakage account, and §8 states, at equal prominence, what none of this promises.

Each pillar section opens with its idea in one breath — an italic express-lane sentence an expert can read alone before jumping to the formal box — and closes with an honest-boundary block at the same prominence as the claim. Every number carries a provenance tag: [verified] means externally recomputable (a textbook value or a closed form restated in the box); [internal, *script*] means it regenerates from the named self-contained script at seed 20260816, shipped with the paper.

2 The design: one work order, two fences, two gates

The work order. Both principals sign the same contract before any key is released:

$$C = \big\langle\, H(\text{data manifest}),\, H(\text{agent bundle}),\, H(\text{runtime}),\, H(\text{policy}),\, \text{function, tools,}\\ \text{recipients, output schemas, budgets, retention, validators, payment, liability}\,\big\rangle.$$

The order names the measured runtime and agent bundle, the encrypted data inputs, the allowed tools and external providers, the result and telemetry schemas, the output-bit/query/timing budgets, the declassification authorities, retention and key destruction, success evidence, dispute procedure, and guarantee. Any change makes a new hash and requires both signatures.

Dual-attested key release. The workload boots in a confidential VM and creates an ephemeral job key with fresh attestation evidence binding runtime, policy, worker image, key, nonce, and epoch. Derek and Erin *independently* verify the attestation; Derek's key broker releases the data key only to the approved measurement, and Erin's key broker releases the model/skill key under the same condition — the attester/verifier/relying-party pattern of IETF RATS [25]. The job then runs with no ambient network or credentials. Neither principal, and not the cloud operator, can read the other's plaintext: the room is sealed from three sides at once.

Two fences. Erin's worker is treated as hostile even if Erin is honest — prompt injection or a compromised dependency can turn a loyal agent into an exfiltrator — so it receives *semantic handles only*: `read`(handle, range), `infer`(model handle, prompt handle), `execute`(tool handle, args), `write`(result handle, bytes), `request_declassification`(candidate handle, channel). An attested in-guest monitor mediates labels, handles, and encrypted content; a Derek-controlled outer gateway mediates network, storage, timing, destination, and volume. Direct outbound TLS is forbidden; permitted services terminate at a policy gateway. And because an ordinary

remote model API is itself a trusted reader of plaintext, the model runs *inside* the confidential domain — an external inference endpoint would be a third principal the contract never admitted.

Whole-worker taint and two gates. After the first secret read the entire worker state, and every later emission, is conservatively tainted until job destruction — decentralized labels in the Myers–Liskov style [7], owned by mutually distrustful principals, under which ordinary computation can only *add* restrictions. This is the same shape as capability attenuation in the program's identity layer, where a delegated token never carries more authority than its parent: authority only shrinks, and every widening is an explicit, witnessed exception rather than a computation. Removing an owner's clause requires a declassification authority delegated by that owner, and the only exits are the two declassification gates — Derek's and Erin's, each executing a declared release function — with exactly three pre-authorized channels: a rich result to Derek; fixed-schema, aggregated or differentially private feedback to Erin; a padded public receipt to both. Each party gets an audience-redacted signed receipt $R = \mathrm{Sign}\big(H(C), H(\text{attestation}), H(\text{inputs}), H(M), H(\text{output ciphertext}), \text{labels}, \text{declassifications}, \text{counters}, \text{nonce}, \text{ep}$ Merkle-anchored in both principals' ledgers. The receipt proves what the measured system reported under a policy; it does not prove the answer is true, nor that no unmodeled side channel existed.

Alongside the four pillars, the design carries a set of deliberately tractable side properties — label monotonicity (no label weakens without a declassification witness), complete mediation (every release has exactly one gate decision), provenance closure (every release traces to committed input, model, runtime, policy, and declassifier nodes), replay safety (each job nonce settles at most once), measurement binding (neither secret key is released for a different runtime or policy), and rollback detectability (accepted epochs strictly increase in both external ledgers). These are design invariants backed by implementation and tests, not theorems; the program's discipline is to label every important statement as theorem, design invariant, model-checked property, or empirical hypothesis, and never let one impersonate another.

Pillar	Name	Claim	Verification	Artifact
I (§3)	Noninterference	silence except through the slot	exhaustive, depth 7; 2/2 mutations	`c1_nonin`
II (§4)	Enforceability	the channel, never the token	product-automaton checker	`b3_contr`
III (§5)	Conservation	the ledger conserves	exhaustive + 2000 random; 2/2 mut.	`a3_epsil`
IV (§6)	Detection	power curve + alarm clock	analytic vs. simulation agreement	`a4_canar`

The four rows are not four independent guarantees stacked for redundancy; they are four links in one pipeline, each closing a gap the others leave open, in exactly the order Figure 1 draws it. Enforceability (II) fixes *where* a boundary can exist at all — gate the channel, or nothing about the design is provably enforceable. Noninterference (I) then proves that boundary, once installed, is silent except through its declared release. Conservation (III) meters the cumulative cost of everything that legitimately crosses it, because a gate honest on every single release can still bankrupt the contract over many releases — an individually-sound decision rule is not a portfolio-sound one. And Detection (IV) polices for what tries to cross without going through the meter at all: a compromised worker, a bug, an adversary who found an unmediated path. Drop any one link and a concrete gap reopens, named inside its box in the figure below; the composed result — an honestly bounded leakage account, not an eliminated one — is §7.

3 Pillar I — Noninterference: silence except through the slot

Run the whole world twice — identical except for Derek's secret — under every possible sequence of moves, and prove Erin's view is bit-for-bit identical whenever the gate's declared function maps the two secrets to the same release — formal statement in the box below.

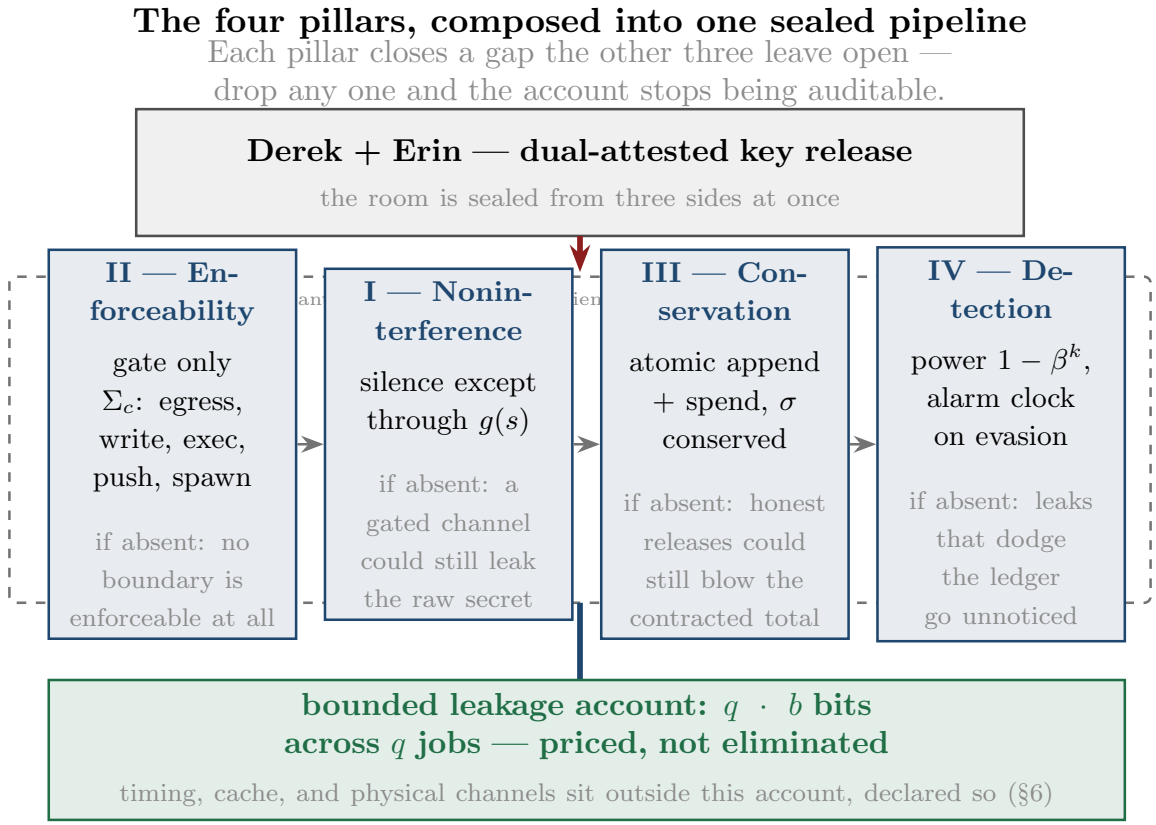

Figure 1: The four pillars composed into the sealed pipeline, in the order the closing section states it: enforceability (II) says where the boundary can be, noninterference (I) says the boundary holds, conservation (III) meters what crosses it, detection (IV) polices evasion. Each box also names the gap left open if that pillar alone were dropped.

Intuition. You cannot inspect a wall for holes by staring at bricks; you inspect it by standing outside in two parallel worlds and asking whether anything you can see ever differs. If Derek's secret is 0 in one world and 2 in the other, and the gate is contracted to release only *parity*, then a sound room makes those worlds indistinguishable to Erin forever, under every interleaving of loads, reads, submissions, releases, and reports. The analogy with teeth is the voting booth: the turnstile clicks once per voter and the tally board shows only party totals; a poll-watcher outside learns exactly the declared aggregate and nothing about any ballot, *no matter how long she watches or in what order voters arrive.* That is noninterference *modulo declassification* in the Goguen–Meseguer lineage [1, 2]: silence except through the slot, and through the slot, only what the contract says. Figure 2 draws the mapping by relations, not surface features. The misread to preempt: the theorem does not say the slot is safe — it says the slot is the *only* opening; whether the declared function g releases too much is a contract question the next two pillars price. Said in the booth's own vocabulary, so the fence arrives while the analogy is still on the ground: the guarantee is about the walls and the turnstile, never about what the board is asked to print. A per-precinct board reporting a precinct with a single voter discloses that voter's ballot exactly, and no property of the booth prevents it — the room stayed sound; the tally was indiscreet. That is where the analogy's fence runs, and it is why Pillars III and IV exist: they price the tally, not the walls.

Theorem 1 (noninterference modulo declassification; finite model, exhaustive). Fix the clean-room transition system with secrets $s \in \{0, 1, 2, 3\}$, Erin-observable state Obs_E, and declared release function $g(s) = s \bmod 2$, *where the gate applies g to committed input state rather than to worker-computed state.* For secrets s, s' and *every* action sequence: if $g(s) = g(s')$ then Erin's observation sequences are identical; if $g(s) \neq g(s')$ they may differ only at explicit gate-release steps.

That italicised clause is a hypothesis, not a modelling convenience, and it is what makes the theorem an instance of *delimited release* [14] in the degenerate case where delimited release collapses to noninterference — their own condition is that no variable used in a declassification is updated before it. §8 records what the hypothesis costs.

Verification. Exhaustive over all interleavings to depth 7 × all secret pairs: zero distinguishing observations in the honest model [internal, `c1_noninterference.py`]. The schedule itself is secret-independent, checked separately — an enabledness difference would itself be a channel. The Rushby unwinding condition *local respect* (Derek-side actions never touch Erin-observable state) holds mechanically [3]. **Mutations:** (m1) a gate that leaks the raw secret is caught distinguishing secrets $(0, 2)$ — an *equal-parity* pair the honest gate provably keeps identical — with witness trace `load` → `read` → `submit` → `release`, after which Erin sees `(gate,0)` in one world and `(gate,2)` in the other; (m2) a worker bypassing the gate is caught in a 3-step trace. Reverting the mutations restores the property [internal, `c1_noninterference.py`].

Numbers by hand. Four secrets give $\binom{4}{2} = 6$ unordered pairs; two of them — $(0, 2)$ and $(1, 3)$ — are equal-parity, and it is exactly those two on which the honest gate must keep Erin's worlds identical forever, and did. The m1 witness is a four-step, human-readable proof that a particular gate breaks its contract; a CISO can read it aloud. The full mutant grid, honest gate against both breaks, is Figure 3. *Now you try:* with $g(s) = s \bmod 2$ over secrets $\{0, \ldots, 7\}$, how many unordered secret pairs must a sound room keep Erin-indistinguishable? (Both parity classes have four members: $2\binom{4}{2} = 12$ pairs.)

What it buys. The product's headline sentence in its corrected, provable form: *every release is explicit, gated, and bounded.* This is the finite-model core of the paper; the general theorem over unbounded state is the Isabelle/AFP unwinding obligation (§9), for which this model is the blueprint — the same three unwinding conditions, discharged mechanically rather than exhaustively.

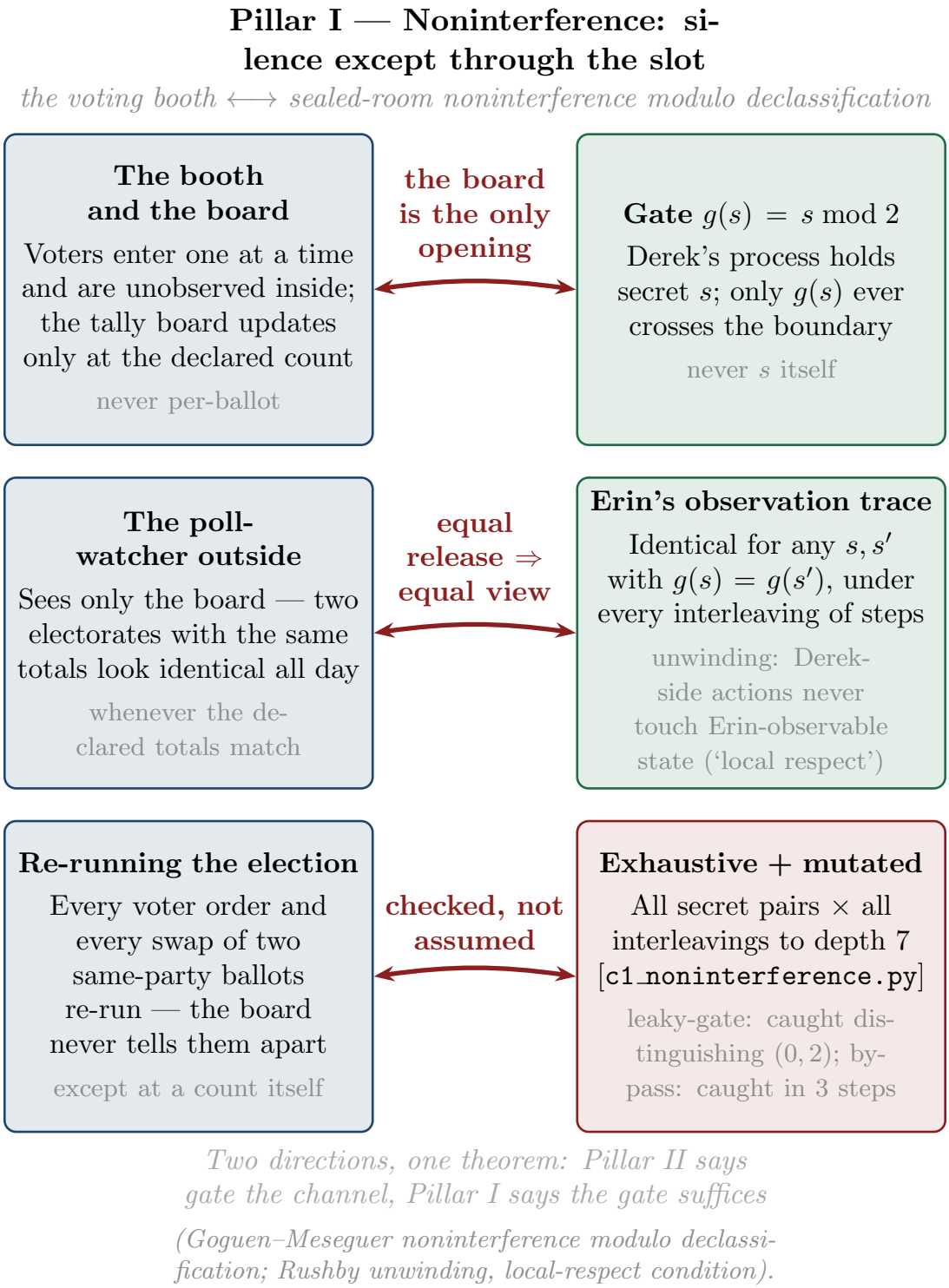

Figure 2: Relation-map for Pillar I. The voting booth maps to the sealed workroom by *relations*, not surface features: booth walls → attested isolation (no observation of the work); the tally board → the declassification gate (the sole declared release); "watching all day reveals no ballot" → two-run observational equivalence under every interleaving. The analogy predicts, correctly, that the residual risk is a too-revealing tally — a per-precinct board with one voter is a leaky g, which is exactly what Pillars III and IV price.

Honest boundary. Finite depth, finite secrets: depth-7 exhaustion is a proof about the model, not the deployment; the lift is §9's job. The guarantee is possibilistic — timing, token counts, and termination channels are out of model, as the design declares (§8). The gate's *specification* is the residual oracle: we prove releases match g; choosing a g that leaks too much is a contract failure no theorem prevents — that risk is priced, not eliminated, by Pillars III–IV.

And there is a second residual, on the other side of the same gate. The theorem quantifies over action sequences of a model whose gate applies g to committed input. It therefore does *not* cover a worker that launders: computing $t = f(s)$ and submitting t so that the gate honestly releases $g(t)$. That is the classical escape-hatch laundering attack, and the delimited-release literature both names it and rules it out by enforcement [14] — an option unavailable to us, because an LLM's release candidate is arbitrary model output rather than a syntactically identifiable expression the checker can quantify over. We therefore price that residual in §7 rather than eliminating it, which is the reason the $q \cdot b$ account exists at all. Our present mutation suite cannot exhibit the attack — the model's `submit` carries no payload — so the suite tests that the gate releases g and nothing else, not that the gate's *argument* is honest; closing that gap is a model extension, listed in §9.

And a human who reads a released parity may still remember it: the system bounds channels, not minds.

4 Pillar II — Enforceability: the channel, never the token

A safety policy over an agent's behavior can be enforced by a hypervisor if and only if it constrains only events the hypervisor mediates — so the enforceable clean-room boundary is the egress channel, and any promise phrased over model-internal events is detect-only forever — formal statement in the box below.

Intuition. A bouncer at the door of a club can prevent anyone from *entering or leaving*; no bouncer can prevent a patron from *thinking about* leaving. Split the agent's event alphabet accordingly: controllable events the mediation layer can veto (Σ_c: `fs_write`, `net_egress`, `exec_tool`, `git_push`, `spawn_child`) and uncontrollable events it can at best observe (Σ_u: `model_emit_token`, `in_context_read`, `internal_plan`). Ramadge–Wonham supervisory control theory [5] then answers, per policy, whether a supervisor achieving exactly that policy exists.

Theorem 2 (enforceability = controllability). Let L be a prefix-closed plant language and let $K \subseteq L$ be a *nonempty, prefix-closed* safety policy over the agent alphabet $\Sigma = \Sigma_c \cup \Sigma_u$. Then K is regimentable (enforceable by the mediation layer) iff K is controllable: $\overline{K}\Sigma_u \cap L \subseteq \overline{K}$ [verified framework, [5]]. Both hypotheses on K are critical and easy to drop by accident: $K = \emptyset$ is vacuously controllable but no supervisor realises it, since every supervised language contains the empty trace. The same nonemptiness condition appears in Basin et al.'s characterisation [17], whose controllable/observable split is the one we use here. Consequences, from the product-automaton checker [internal, `b3_controllability.py`]: *forbid egress / push / write / exec / spawn* ⇒ regimentable; *forbid emit-token / context-read / internal-plan* ("the confident lie") ⇒ detect-only forever. The decisive compound case — *"no `net_egress` after `in_context_read` of a secret"* — is **regimentable**: the policy permits the uncontrollable read, records it as taint, and gates only the controllable egress.

The compound case is the whole clean-room design in one line, and it is a computational vindication of the architecture correction in §1: gating the uncontrollable trigger (the model

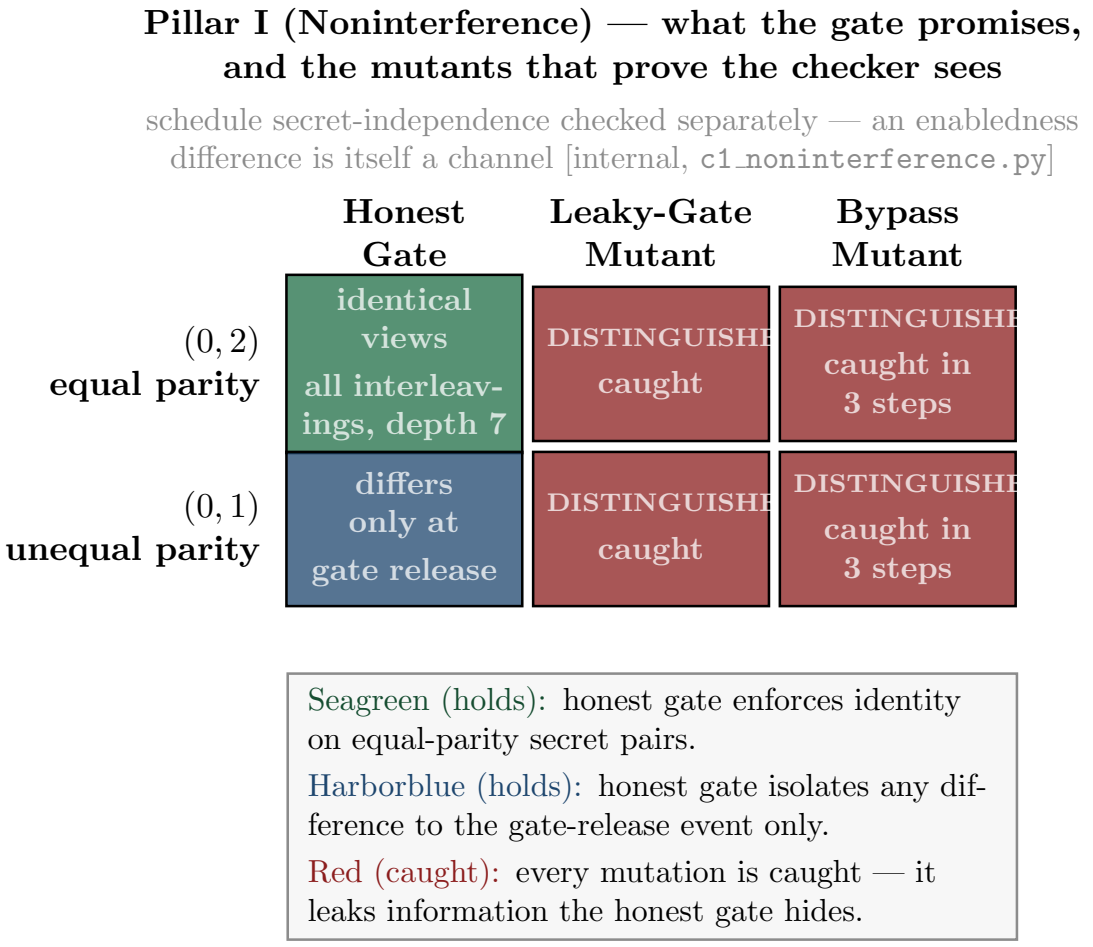

Figure 3: Regime diagram for Pillar I: what the gate promises, and the mutants that prove the checker sees. The property holds on the shaded region (equal declassified value, any interleaving); the two mutation witnesses (leaky gate; gate bypass) sit provably outside it, each with a concrete trace [internal, `c1_noninterference.py`].

reading, thinking, emitting) would be unregimentable, while gating the controllable effect (the egress channel) is not. Whole-worker taint is exactly what the theorem licenses: granting the unsoundness premise stated in §1 — inherited here as an empirical hypothesis, not re-asserted — the sound over-approximation is to taint *everything* after the first secret read and spend all enforcement budget on the one boundary where enforcement is possible. This is why the honest claim — every release explicit, gated, bounded — *replaces* "mathematically cannot phone home": the latter quantifies over uncontrollable events and is therefore not a theorem about any implementable supervisor; the former quantifies over the mediated channel and is Theorem 1.

What it buys. Pillars I and II are the same boundary proved from two independent directions: controllability says the clean room *must* gate the channel and not the token (no alternative design is enforceable); two-run equivalence says gating the channel *suffices* for the contracted secrecy. Together they close the design question. The taint-drop-egress mechanism of the naive pitch survives demoted to what it really is: a containment layer beneath the gate — the "confine" rung of the risk ladder — not the guarantee.

Honest boundary. Controllability is relative to the mediated alphabet: partial observation shrinks the regimentable set (Lin–Wonham [6]), and policies referencing unmediated *state* need that extension. Detect-only is not hopeless — it is a different business: post-hoc audit and slashing, which this program treats elsewhere (Paper 3). And complete mediation — every effect passes through the layer — is the deployment assumption the two remaining pillars inherit; it is an attested property of the workroom, not a theorem.

5 Pillar III — Conservation: the budget is a ledger, and the ledger conserves

Make the privacy budget a ledger — one atomic transition appends the record and adds the spend — and conservation of $\sigma = \sum_\Lambda \varepsilon_i$ survives every interleaving, because the single writer makes every concurrent history observationally sequential — formal statement in the box below.

The scene, resumed. Derek's data never leaves the room; what leaves is a stream of gated releases, each carrying a differential-privacy cost ε_i. The contract says the total never exceeds $\varepsilon_{\max}$. But releases race: two of Erin's jobs invoke the gate concurrently, and the classic time-of-check/time-of-use gap between "is there budget?" and "spend it" is exactly where a meter drifts from its journal. The intuition is double-entry bookkeeping: balance and journal updated in one stroke, so an auditor summing the journal always reproduces the balance.

Theorem 3 (ε-conservation). State $= (\sigma, \Lambda)$ with Λ append-only. The only spending transition is release(ε_i): if $\sigma + \varepsilon_i \le \varepsilon_{\max}$ it *atomically* appends $(\varepsilon_i, \text{artifact hash}, \text{policy hash})$ to Λ and sets $\sigma \mathrel{+}= \varepsilon_i$; otherwise it refuses and changes nothing. Then every reachable state satisfies $\sigma = \sum_\Lambda \varepsilon_i$ and $\sigma \le \varepsilon_{\max}$ — including under concurrent invocation, since the kernel's single-writer serialization makes any concurrent execution observationally equal to some sequential interleaving, over which the induction runs unchanged (base: $\sigma = 0, \Lambda = ()$; step: the guard preserves the bound, the atomic pair preserves the equality; non-spending transitions touch neither).

Corollary (what the conserved sum means). The ledger is a *privacy filter* in the sense of Rogers, Roth, Ullman and Vadhan [18]: a stopping rule that halts before a pre-specified budget is exceeded (their Definition 3.1). The distinction matters here, because our ε_i are requested adaptively and may differ between releases, and the textbook sequential-composition theorem fixes the privacy parameters in advance. What licenses $\sigma \le \varepsilon_{\max} \Rightarrow (\varepsilon_{\max}, 0)$-DP under adaptive heterogeneous spend is their Theorem 3.4 — that

summing *realised* parameters is a valid filter — not sequential composition.

Why not just quote the advanced bound. The same authors show that substituting the advanced-composition bound into such a filter is *not* valid: the natural heuristic "run the DRV bound on whatever the ledger actually spent" provably breaks when the parameters or the stopping point are adaptively chosen. We therefore quote the advanced bound $\left(\sqrt{2k\ln(1/\delta')}\,\varepsilon + k\varepsilon(e^{\varepsilon}-1),\ \delta'\right)$ [9] *only* for engagements in which k and the per-release ε are fixed in the work order — where the schedule is non-adaptive and DRV applies directly — and use the adaptively valid filter of Whitehouse et al. [19], which recovers the advanced-composition rate at worse constants, otherwise. Getting this wrong is not conservative: it certifies a smaller ε than the run is entitled to.

Who the certificate binds, and when it says nothing. The DP reading of σ presupposes that each recorded release *is* an ε_i-DP mechanism. Against the malicious worker of §7 — one choosing b bits adversarially — that presupposition fails outright: such a worker is not running a DP mechanism, and its recorded ε_i is a number it declared about itself. In that regime $\sigma \le \varepsilon_{\max}$ certifies nothing, and the $q \cdot b$ min-capacity account of §7, not the ledger, is the operative bound. The two accounts are for two different adversaries and we own both: systems that schedule DP budgets across concurrent jobs make the corresponding assumption explicit — the analyst is trusted to implement DP correctly [21] — and we decline that assumption about Erin's worker, which is precisely why we cannot lean on the ledger alone. Stating the two in one currency needs the ε-to-min-entropy correspondence of Alvim et al. [23], which we do not discharge here.

Verification (Theorem 3). Exhaustive explicit-state check, two concurrent clients racing programs $(2,2)$ and $(1,2)$ against $\varepsilon_{\max} = 4$: all 15 reachable states satisfy both invariants under every interleaving [internal, `a3_epsilon_ledger.py`]. The state count is order-sensitive — a denied release still advances the client's program counter, so distinct commit orders are distinct states; as a multiset the answer would be 11. **Mutations:** (m1) spend-without-append breaks conservation in 1 step; (m2) the torn write (log ε, add $\varepsilon - 1$) breaks it in 1 step *and* in 3 steps drives the *recorded* log to $\sum_\Lambda = 5 > \varepsilon_{\max} = 4$ — the undercounting balance silently admits auditable over-spend, so atomicity is what makes the budget promise auditable [internal].

Numbers by hand. At $\delta' = 10^{-6}$: for k=32 releases at ε=0.1, basic composition gives $32 \times 0.1 = 3.20$ while advanced gives 3.31 — not yet paying; at k=128, ε=0.05, basic gives 6.40 but advanced gives 3.30 — paying [verified]. The crossover tells the contract writer exactly which accounting to quote at which engagement length: below it, quote the exact, δ-free sequential bound; above it, the $\sqrt{k}$-scaled advanced bound (Figure 4). *Now you try:* at k=8, ε=0.5, which bound is smaller? (Basic: 4.00; advanced: 10.03 — quote sequential, and by a wide margin [verified].)

What it buys. The clean-room contract's budget line becomes an auditable invariant with the same proof shape as the harbor's bond-conservation result, and the mutation suite shows the invariant has teeth: both canonical atomicity breaks are caught with shortest traces, and the torn write's deeper crime — recorded over-spend past the contracted maximum — is exhibited concretely.

Honest boundary. Conservation governs *recorded* spend; that all spend is recorded is precisely the complete-mediation assumption — Pillar II is the prerequisite, not a nicety. Advanced composition pays only past the k-crossover. And the theorem conserves the meter, not the meaning: choosing per-release ε_i honestly is the DP mechanism's obligation, outside this ledger — differential privacy bounds individual influence in aggregates; it does not protect the whole dataset, nor Erin's model IP.

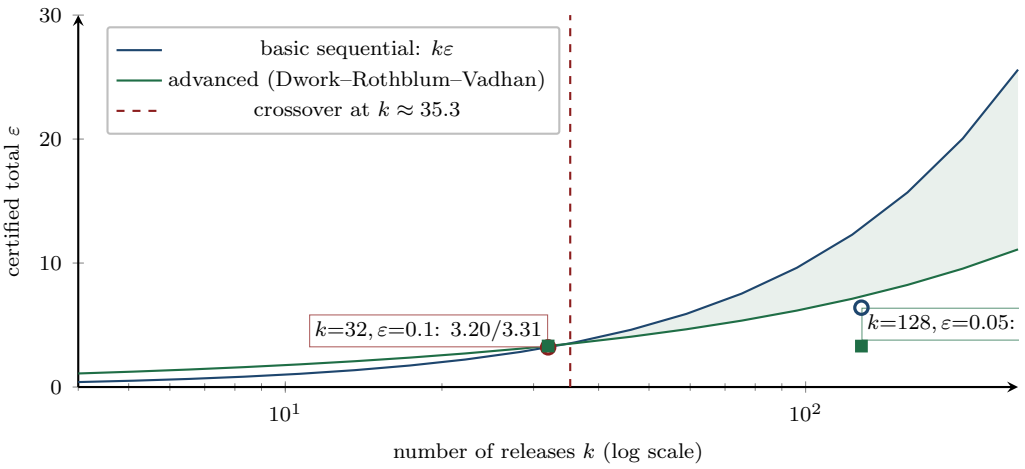

Figure 4: Regime diagram for Pillar III: which DP accounting to quote at which engagement length ($\delta' = 10^{-6}$). Below the crossover in k the sequential bound $k\varepsilon$ is exact, δ-free, and smaller; past it, advanced composition's $\sqrt{k}$ scaling wins. The session's two measured points (k=32, ε=0.1 and k=128, ε=0.05) are marked [verified; regenerated by `a3_epsilon_ledger.py`].

6 Pillar IV — Detection: canaries with a power curve and a clock

Independent per-canary misses turn k smuggled canaries into detection probability $1 - \beta^k$; uniform planting turns leak size into a hypergeometric operating curve; and Wald's sequential test against the permutation null turns leak intensity into an expected time-to-detection — formal statements in the box below.

Intuition. Dye packs in bank bundles: each pack fails alone with probability β, so grabbing k dyed bundles fails to stain with probability β^k; how many dyed bundles a thief grabs is set by how much he takes and how densely packs are planted. The sequential half is quality control: each gate output is a Bernoulli trial, and the sequential probability ratio test is the optimal stopping rule for "is this stream leaking?" Pillars I–III govern what the gate is *allowed* to pass; this pillar measures what happens when something tries to pass anyway — "we have canaries" is a vibe, and a CISO buys a number.

Theorem 4a (power). With per-canary false-negative rate β, independent across canaries, a leak carrying k canary spans is detected with probability $1 - \beta^k$. Under uniform planting of c canaries in an n-span corpus, a leak of m spans carries $K \sim \text{Hypergeom}(n, c, m)$ canaries, so $\Pr(\text{detect} \mid m) = 1 - \mathbb{E}[\beta^K] \approx 1 - \left(1 - \frac{c}{n}(1-\beta)\right)^m$ — the operating curve.

Theorem 4b (latency, Wald). Gate outputs are Bernoulli with canary-hit rate p_0 under the no-leak permutation null and $p_1 > p_0$ under exfiltration. The SPRT with error targets (α, γ) — false-alarm target α, miss target γ — stops, under leak, after expected $\mathbb{E}_1[N] = \left[(1-\gamma)\ln\frac{1-\gamma}{\alpha} + \gamma\ln\frac{\gamma}{1-\alpha}\right] / \mathrm{KL}(p_1 \| p_0)$ outputs (analogously $\mathbb{E}_0[N]$ under the null), and is optimal in the Wald–Wolfowitz sense [10, 11]. We write γ rather than the customary β or b: β is already the per-canary miss rate of Theorem 4a, and b is the release-channel width of §7, the paper's headline quantity.

Verification. n=10^4, c=100 (density 1%), β=0.2: a 100-span leak is caught with probability 0.554, an 800-span leak at 0.999; exact hypergeometric, binomial approximation, and simulation agree to Monte-Carlo precision across the curve, and the $1-\beta^k$ line is confirmed at k=1, 2, 5 [internal, `a4_canary_sprt.py`]. SPRT at p_0=10^{-3}, p_1=10^{-2}, α=0.01, γ=0.05:

Wald $\mathbb{E}_1[N] = 297$ vs simulated 359 (the gap is boundary overshoot, which the Wald identities ignore — and overshoot only *helps* the error rates: realized miss $0.048 \leq 0.05$, false alarm $0.005 \leq 0.01$); $\mathbb{E}_0[N] = 432$ vs simulated 438. Latency sweep: 10× leak intensity cuts time-to-detection ≈ 30× [internal].

Numbers by hand. At $\beta = 0.2$, two smuggled canaries are caught with probability $1 - 0.04 = 0.96$ [verified]; the full operating curve against leak size is Figure 5. *Now you try:* at the same β, what does a five-canary leak survive with? ($\beta^5 = 0.00032$ — detection 0.9997 [verified].)

Pillar IV (Detection) — the operating curve a CISO can buy, and the boundary that voids it

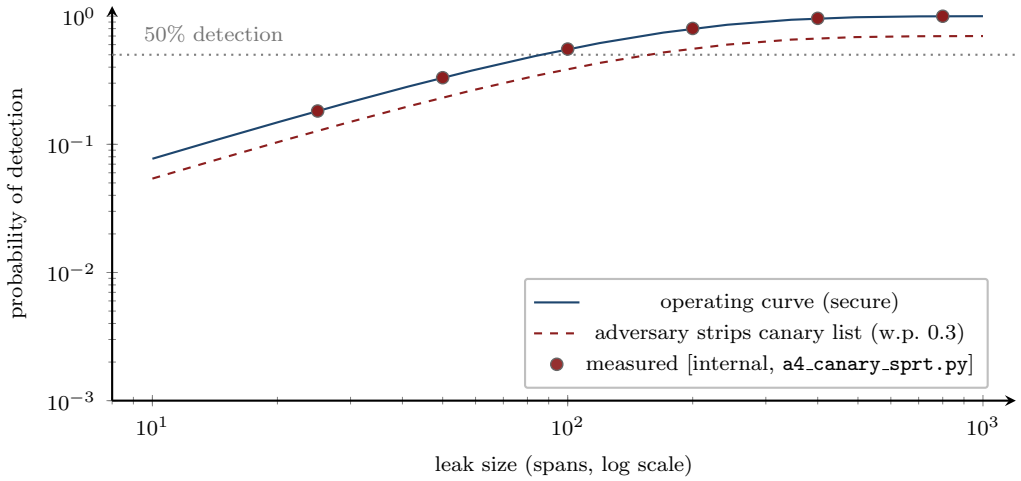

Figure 5: Regime diagram for Pillar IV: the operating curve a CISO can buy, and the boundary that voids it. Left: Pr(detect) against leak size m at density 1%, $\beta = 0.2$ — exact, approximate, and simulated agree [internal, `a4_canary_sprt.py`]. The guarantee's critical regime is secret, uniform planting: an adversary who obtains the canary list with probability 0.3 and strips all of them drops measured power from 0.9997 to 0.698 at k=5 [internal].

What it buys. The clean-room contract can *quote* Pr(detect) = f(leak size) and expected alarm latency per leak intensity — so the exfiltration bond is re-scoped to its honest job, funding detection and response rather than pretending to cover unbounded breach damage.

Honest boundary. Independence and *secrecy* of planting are critical: an adversary who obtains the canary list with probability 0.3 and strips all of them drops measured power from 0.9997 to 0.698 at k=5 [internal, `a4_canary_sprt.py`] — canary secrecy is part of the security boundary, priced explicitly. β is a measured property of the suppressor, not an assumption. The Wald stopping-time formulas are approximations (overshoot inflates $\mathbb{E}[N]$ ∼20% here, conservatively); and the SPRT detects *statistical* leak signatures — a single-shot exfiltration is Theorem 4a's job, not 4b's.

7 The leakage budget: $q \cdot b$ bits, before timing

The four pillars compose into one information-theoretic account, and the account is deliberately unromantic. It is also where the paper pays the three debts it took on earlier, and it is worth naming them rather than discharging them silently: the laundering residual left open by Theorem 1's committed-input hypothesis (§3), the malicious worker against whom $\sigma \leq \varepsilon_{\max}$ certifies nothing (§5), and the arbitrary-telemetry channel disclaimed in §8. All three are the same debt seen from three sides, and it has one price. If an Erin-visible channel permits b freely chosen

bits per job, a malicious worker — one that launders, computing $t = f(s)$ and submitting t so the gate honestly releases $g(t)$ — can exfiltrate up to b bits through it: the noninterference theorem guarantees only that those bits pass through the declared gate, not that they are innocent. Across q jobs the raw capacity is $q \cdot b$ bits; choosing among s timing slots adds up to $\log_2 s$ bits per event, which is why timing sits outside the possibilistic model and inside §8. In Geoffrey Smith's vocabulary [8], the design does not drive min-entropy leakage to zero; it makes leakage a *declared, metered* quantity: fixed schemas, padding, coarse statuses, query limits, and scheduled release turn an unlimited covert channel into a leakage budget the contract states, the ledger of Pillar III meters (with DP composition giving the aggregate-feedback portion its formal semantics), and the canaries of Pillar IV police for evasion around.

The design rule that falls out is quotable: *never negotiate over "does this output contain a secret" — negotiate over channel capacity.* The first question is undecidable in practice for prose; the second is arithmetic. A contract that grants Erin a 64-bit status schema per job, 200 jobs, has granted at most 12,800 bits of adversarial capacity before timing — and can price, bond, and canary-police exactly that.

8 What cannot honestly be promised

The pitch discipline of this paper is that the boundary list is as prominent as the theorem list. None of the following is promised, and a contract that implies any of them is overclaiming:

- **Zero model leakage with unlimited useful adaptive outputs to Derek.** The model-confidentiality property is necessarily weaker than the data property, because Derek *intentionally observes* evaluations of M: unlimited adaptive black-box access enables model extraction. Erin's confidentiality is stated relative to the contracted query-and-output interface, not as absolute secrecy — the asymmetry is structural, not an engineering gap.
- **Zero data leakage with arbitrary natural-language telemetry to Erin.** That channel is the $q \cdot b$ budget of §7; "zero" is available only at $b = 0$.
- **Elimination of timing, cache, speculative, firmware, or physical channels.** The noninterference guarantee is possibilistic; these channels are out of model, declared so, and partially mitigated (padding, scheduled release) rather than closed.
- **Confidentiality through an unapproved external model or tool provider.** A remote API that reads plaintext is another trusted reader; the model runs inside the confidential domain or the guarantee does not apply.
- **Correctness of the answer.** The receipt proves what the measured system reported under a policy; it does not prove the financial advice was good.
- **Literal deletion of information a party already observed**, availability if a key owner refuses or aborts, or security against colluding hardware vendor, cloud operator, and principal — each needs a strictly stronger threat model (MPC/FHE backends are the high-assurance option for narrow fixed computations, not the first implementation for arbitrary tool-using agents).
- **Bounds on minds.** A human who reads a released aggregate may remember it; the system bounds channels.

9 The lift: from exhaustive model to Isabelle unwinding

Theorem 1 is exhaustive on a finite model; the general claim over unbounded state is a proof obligation with a known shape. Rushby's channel-control formulation [3] reduces noninterference

for a transition system to three local *unwinding conditions* — output consistency, step consistency, and local respect — each a per-transition lemma amenable to mechanization; the Isabelle Archive of Formal Proofs already carries stateful intransitive-noninterference and noninfluence developments in exactly this lineage. The finite model of §3 is the blueprint: its local-respect check (Derek-side actions never touch Erin-observable state) already holds mechanically, its declassification steps are the intransitive "downgrader" domain of Rushby's framework, and the lift consists of restating the clean-room transition relation abstractly and discharging the three lemmas once, for all depths and all secrets. We flag the lift as the program's stated next obligation for this paper rather than folding a half-finished mechanization into it; the finite result plus mutation suite is what the lift will certify at scale, not a substitute for it.

One target-property precision the lift must get right from the start: Rushby's own soundness result for these unwinding conditions is with respect to Haigh–Young intransitive purge, but van der Meyden shows the conditions are sound *and complete* for the strictly stronger TA-security property instead — which additionally preserves the *order* of permitted transmissions, not just which actions survive intransitive purging [4]. Discharging the three lemmas therefore certifies TA-security, not the weaker purge-based notion; the lift should state its target as TA-security explicitly rather than assume the weaker property falls out for free.

10 Related work, and what is actually new

Imported. Noninterference and its unwinding proof technique are Goguen–Meseguer [1, 2]; the intransitive/channel-control generalization that models a declassifier as a distinguished domain is Rushby [3]. The enforceability boundary is Ramadge–Wonham supervisory control [5] with the Lin–Wonham partial-observation refinement [6]. Labels owned by mutually distrustful principals are the Myers–Liskov decentralized label model [7]. Quantitative leakage is Smith's min-entropy framework [8]; the sequential test is Wald [10, 11]; attestation vocabulary is IETF RATS [25].

Pillar I's property has a name in the language-based literature and we adopt it rather than reinvent it: Theorem 1's first clause is *delimited release* [14], whose own running example is the release of a secret's parity, and its second clause is the possibilistic shadow of *gradual release* [15]. On the Sabelfeld–Sands taxonomy [16] the ε-release ledger is a *what* policy (g) with a *when* policy (the budget gate) and a partial *who* policy (per-owner declassification authority); we make no *where* claim. Pillar II's alphabet split is likewise not ours: Basin, Jugé, Klaedtke and Zălinescu [17] distinguish controllable from merely observable actions and characterise enforceability relative to that distinction, and their nonemptiness condition is a hypothesis Theorem 2 should carry. Pillar III's object is a *privacy filter* [18], realised as a runtime budget in PINQ [20] and scheduled across concurrent jobs in PrivateKube [21]; the combination of information-flow control with a differential-privacy price on the declassifier is Birgisson, McSherry and Abadi's proposal [22].

And the design itself is not ours. Ryoan [24] already confines a proprietary module inside an attested enclave so that a data owner and a code owner who distrust each other both keep their secrets, with per-principal declassification, padded fixed-size outputs, controlled explicit and covert channels, and — in its §5.2 — the same $\log_2 s$ bit count for a timing channel with s statically determined slots, and the same observation that repeating a request accumulates leakage until the input is exhausted. Our $q \cdot b$ contract restates their arithmetic for a multi-job engagement. Table 1 draws the line row by row, so that a reader who knows Ryoan can see in one glance what is and is not claimed, and a reader who does not gets a working description of it alongside the delta.

New, honestly. No component theorem above is ours, and neither is the confined mutually-distrustful enclave design. What is ours is the adaptation of that design to *tool-using LLM agents*, and the assurance schedule the adaptation forces. Ryoan buys its confinement with a

	Ryoan (2016) [24]	The Sealed Harbor
Workload	one-shot module: sees its input once, keeps no state	looping agent: tools, sub-steps, retained context
How leakage is bounded	*structurally* — there is no retained state to leak from	*metered* — declared channel width plus the ε-ledger
Taint granularity	module confinement	whole-worker taint after the first secret read
Model access	the module runs inside the enclave	the model runs inside the confidential domain; no external inference endpoint
Multi-job accounting	per request; repetition noted as exhausting the input	$q \cdot b$ across q jobs, written into the work order
Evasion	not in scope	canary operating curve + SPRT clock (Pillar IV)
Timing	$\log_2 s$ bits over s static slots (their §5.2)	the same arithmetic, restated per engagement

Table 1: The same confinement idea, priced for a different workload: every row where the two designs agree is Ryoan's, and the budget and detector exist only because a looping agent cannot pay the statelessness price that made Ryoan's bound structural.

request-oriented model in which a module sees its input once and keeps no state; an agent that loops, calls tools, and orchestrates sub-steps cannot pay that price. We replace the structural prohibition with a metered one — which is why a budget contract and a detection theory are critical here and absent there — and we justify whole-worker taint not by engineering tractability but by the unsoundness claim stated in §1 (an empirical hypothesis, not a theorem): token-level taint through a generative model forces a choice between marking every output token and missing semantic flows, so the sound over-approximation is the worker. The four-pillar schedule (II says where the boundary can be, I that it holds, III what crosses it, IV what evades it) has no counterpart in the confidential-computing literature. The correction this delivers to practice is the replacement of "taint analysis prevents exfiltration" and "mathematically cannot phone home" with the claim that is weaker in form and true: *bring the encrypted agent to the encrypted data, and every release is explicit, gated, and bounded.*

Reproducibility

Every [internal] number regenerates from three self-contained Python scripts at seed 20260816, shipped alongside this paper: `c1_noninterference.py` (Pillar I: exhaustive depth-7 check, schedule-independence check, unwinding check, 2/2 mutations caught with witness traces), `a3_epsilon_ledger.py` (Pillar III: 15-state exhaustive interleaving check, 2000-instance randomized sweep, 2/2 mutations with shortest crimes, DP composition table), and `a4_canary_sprt.py` (Pillar IV: $1 - \beta^k$ at k=1, 2, 5, hypergeometric/binomial/simulated operating curve, SPRT latency and error realization, canary-stripping boundary experiment). Pillar II's classification table regenerates from `b3_controllability.py`. Every [verified] number is a closed form the reader can recompute from the formulas in the boxes.

References

[1] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy (Oakland)*, pp. 11–20, 1982.

[2] J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symposium on Security and Privacy*, 1984.

[3] J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report CSL-92-02, SRI International, December 1992.

[4] R. van der Meyden. What, indeed, is intransitive noninterference? (extended abstract). In *Proc. European Symposium on Research in Computer Security (ESORICS)*, LNCS 4734, pages 235–250, 2007.

[5] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[6] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988.

[7] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *Proc. 16th ACM Symposium on Operating Systems Principles (SOSP)*, 1997.

[8] G. Smith. On the foundations of quantitative information flow. In *Proc. FoSSaCS 2009*, LNCS 5504, pp. 288–302, 2009.

[9] C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *Proc. 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 51–60, 2010.

[10] A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[11] A. Wald and J. Wolfowitz. Optimum character of the sequential probability ratio test. *Annals of Mathematical Statistics*, 19(3):326–339, 1948.

[12] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.

[13] A. Russo and A. Sabelfeld. Dynamic vs. static flow-sensitive security analysis. In *Proc. 23rd IEEE Computer Security Foundations Symposium (CSF)*, pages 186–199, 2010.

[14] A. Sabelfeld and A. C. Myers. A model for delimited information release. In *Software Security — Theories and Systems (ISSS 2003)*, pages 174–191. Springer, 2004.

[15] A. Askarov and A. Sabelfeld. Gradual release: unifying declassification, encryption and key release policies. In *Proc. IEEE Symposium on Security and Privacy*, pages 207–221, 2007.

[16] A. Sabelfeld and D. Sands. Declassification: dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.

[17] D. Basin, V. Jugé, F. Klaedtke, and E. Zălinescu. Enforceable security policies revisited. *ACM Transactions on Information and System Security*, 16(1):1–26, 2013.

[18] R. Rogers, A. Roth, J. Ullman, and S. Vadhan. Privacy odometers and filters: pay-as-you-go composition. In *Advances in Neural Information Processing Systems 29*, pages 1921–1929, 2016.

[19] J. Whitehouse, A. Ramdas, R. Rogers, and Z. S. Wu. Fully-adaptive composition in differential privacy. In *Proc. 40th International Conference on Machine Learning (ICML)*, PMLR 202, pages 36990–37007, 2023.

[20] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 19–30, 2009.

[21] T. Luo, M. Pan, P. Tholoniat, A. Cidon, R. Geambasu, and M. Lécuyer. Privacy budget scheduling. In *Proc. 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 55–74, 2021.

[22] A. Birgisson, F. McSherry, and M. Abadi. Differential privacy with information flow control. In *Proc. 6th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, pages 1–6, 2011.

[23] M. S. Alvim, M. E. Andrés, K. Chatzikokolakis, and C. Palamidessi. On the relation between differential privacy and quantitative information flow. In *Proc. ICALP 2011*, LNCS 6756, pages 60–76. Springer, 2011.

[24] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: a distributed sandbox for untrusted computation on secret data. In *Proc. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 533–549, 2016.

[25] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan. Remote ATtestation procedureS (RATS) architecture. RFC 9334, IETF, January 2023.