The Sealed Harbor:

Mutually Confidential Computation with Explicit, Gated, Bounded Releases

Erich Owens

Curiositech LLC

engineering@portdaddy.dev

September 2026
Version 1.0 (textbook edition)

1 Two owners, one answer

Express lane: a sealed room lets an agent compute on data its owner will not hand over, and the room's promise is not "nothing leaks" but "every release is explicit, gated, and bounded"; the four claims that make that promise provable are Theorems 3.1, 5.1, and 6.1, plus the controllability theorem of The Single-Writer Kernel*, and the leakage they leave is the account of Section 7.*

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 standoff with lawyers and air-gapped laptops. The question this chapter answers is whether a *computational* contract can replace the lawyers' one, and what, precisely, it can promise. The systems literature answered a narrower version first: Ryoan [18] 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. Ryoan buys its confinement with a module that sees its input once and keeps no state. A looping, tool-calling agent cannot pay that price, which is why this chapter needs a budget and a detector where Ryoan needed neither. Table 1 is the whole difference in one place.

Table 1: How the sealed room differs from Ryoan. Ryoan confines a stateless module that sees its input once; the sealed room confines a looping agent, so it needs a priced exit and a detector where Ryoan needed neither.

	Ryoan	The sealed room
Workload shape	a request-oriented module, input seen once, no persistent state	a looping, tool-calling agent with state across steps
How leakage is bounded	confinement in the enclave plus fixed-size outputs	explicit gated releases, priced by an ε-ledger and a $q \cdot b$ bit budget
Taint granularity	the whole module	the slot: every release is a declassification event
Model access	the module's code is sealed; no query interface	the model runs inside the domain and is observed only through the contracted query and output interface
Multi-job accounting	none; each request stands alone	the ledger sums spend across jobs and refuses the overdraw
Evasion	prevented by construction for its workload	caught by canaries with a power curve and a Wald clock
Timing	out of scope	declared out of model; padded and scheduled, not closed

The pitch that must never ship. The tempting design, and the one an early product review of this program actually wrote, is a "silicon-enforced NDA": tag Erin's agent the moment it reads Derek's data, drop any tainted egress packet, and declare that the data "mathematically cannot phone home." The first job of this chapter is to say why that sentence is false, and the second is to prove the sentence that replaces it.

Empirical hypothesis 1.1 Token-level taint through a language model is not soundly definable. A taint tracker over a generative model faces a binary choice: mark every output token as tainted by everything the model read, which is useless, or miss semantic flows, which is unsound. The model taints everything it writes with everything it read. This is a design rationale, not a theorem, and it is the one load-bearing statement in this chapter that carries no proof and no script. It rests on the well-known static-analysis result beneath it, that certifying implicit flows soundly forces exactly this dichotomy: the label creep of the Denning certification lattice [11] and the extra restrictiveness a sound purely dynamic monitor must accept relative to a static one [12]. No sound token-granularity flow analysis for a generative model is known; the burden of exhibiting one falls on any design that assumes it.

A secret can ride out in wording, punctuation, error messages, output length, or timing; no regular expression, scanner, or second model 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 architecture therefore controls the *channel*, not the token. It converts the question a security officer will actually ask, *how do I know the wall has no other holes?*, into four provable claims: the gate is the only hole (Section 3); the gate sits on the only boundary that can be enforced at all (Section 4); what passes through it is metered by a conserved budget (Section 5); and what tries to sneak around the meter is caught with quotable probability and latency (Section 6). Section 7 composes the four into a single leakage account, and Section 8 states, at equal prominence, what none of this promises.

2 One work order, two fences, two gates

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

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

Read the tuple in order. The four hashes pin the data manifest, the agent bundle, the measured runtime, and the policy; function and tools fix what may run and what it may call; recipients and output schemas fix who may receive a result and in what shape; budgets bound output bits, queries, and time; retention fixes key destruction; validators name the success evidence and the dispute procedure; payment and liability are the guarantee. Any change makes a new hash and requires both signatures. It is a work unit in the sense of *The Single-Writer Kernel*: the case file that outlives every process that touches it.

Dual-attested key release. The workload boots in a confidential virtual machine 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 and skill key under the same condition, the attester, verifier, and relying-party pattern of IETF RATS [19]. 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, because prompt injection or a compromised dependency can turn a loyal agent into an exfiltrator, so it receives *semantic handles only*:

$$\texttt{read}(\text{handle}, \text{range}) \quad \texttt{infer}(\text{model handle}, \text{prompt handle}) \quad \texttt{execute}(\text{tool handle}, \text{args})\\ \texttt{write}(\text{result handle}, \text{bytes}) \quad \texttt{request\_declassification}(\text{candidate handle}, \text{channel})$$

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. Because a 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 tainted until job destruction: decentralized labels in the Myers–Liskov style [6], owned by mutually distrustful principals, under which ordinary computation can only *add* restrictions. This is the same shape as capability attenuation in *The Anchor Protocol*, 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 receives an audience-redacted signed receipt, Merkle-anchored in both principals' ledgers:

$$R = \text{Sign}\big(\, H(C),\ H(\text{attestation}),\ H(\text{inputs}),\ H(M),\ H(\text{output ciphertext}),\\ \text{labels, declassifications, counters, nonce, epoch}\,\big).$$

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.

Design invariant 2.1 The room's 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. Rollback detectability: accepted epochs strictly increase in both external ledgers. Each is intended to hold and is backed by the implementation and its tests; none is a theorem, and this chapter never lets one stand in for one.

Four links, one pipeline. The four claims are not four independent guarantees stacked for redundancy; they are four links in one pipeline, each closing a gap the others leave open. Enforceability fixes *where* a boundary can exist at all: gate the channel, or nothing about the design is provably enforceable. Noninterference proves that boundary, once installed, is silent except through its declared release. Conservation 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. Detection 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; the composed result, a bounded leakage account rather than an eliminated one, is Section 7. Table 2 is the evidence schedule: each link names its claim, how the claim was verified, and the script that regenerates the verification.

Table 2: The four links of the sealed room and the evidence behind each. The four-pillar claim is an evidence schedule, not a slogan: every row names a regenerable check.

Link	Claim	Verification	Script
Noninterference	silence except through the slot	exhaustive to depth 7; both mutations caught	`c1_noninterference.py`
Enforceability	the channel, never the token	product-automaton checker	`b3_controllability.py`
Conservation	the ledger conserves	exhaustive and 2000 random instances; both mutations caught	`a3_epsilon_ledger.py`
Detection	power curve and alarm clock	analytic against simulation	`a4_canary_sprt.py`

Figure 1 draws this pipeline: one work order, two fences, two gates, and which of the four claims each part produces. The figure orders the links as they compose at run time, enforceability first; the sections that follow take noninterference first because it is the promise the other three exist to keep.

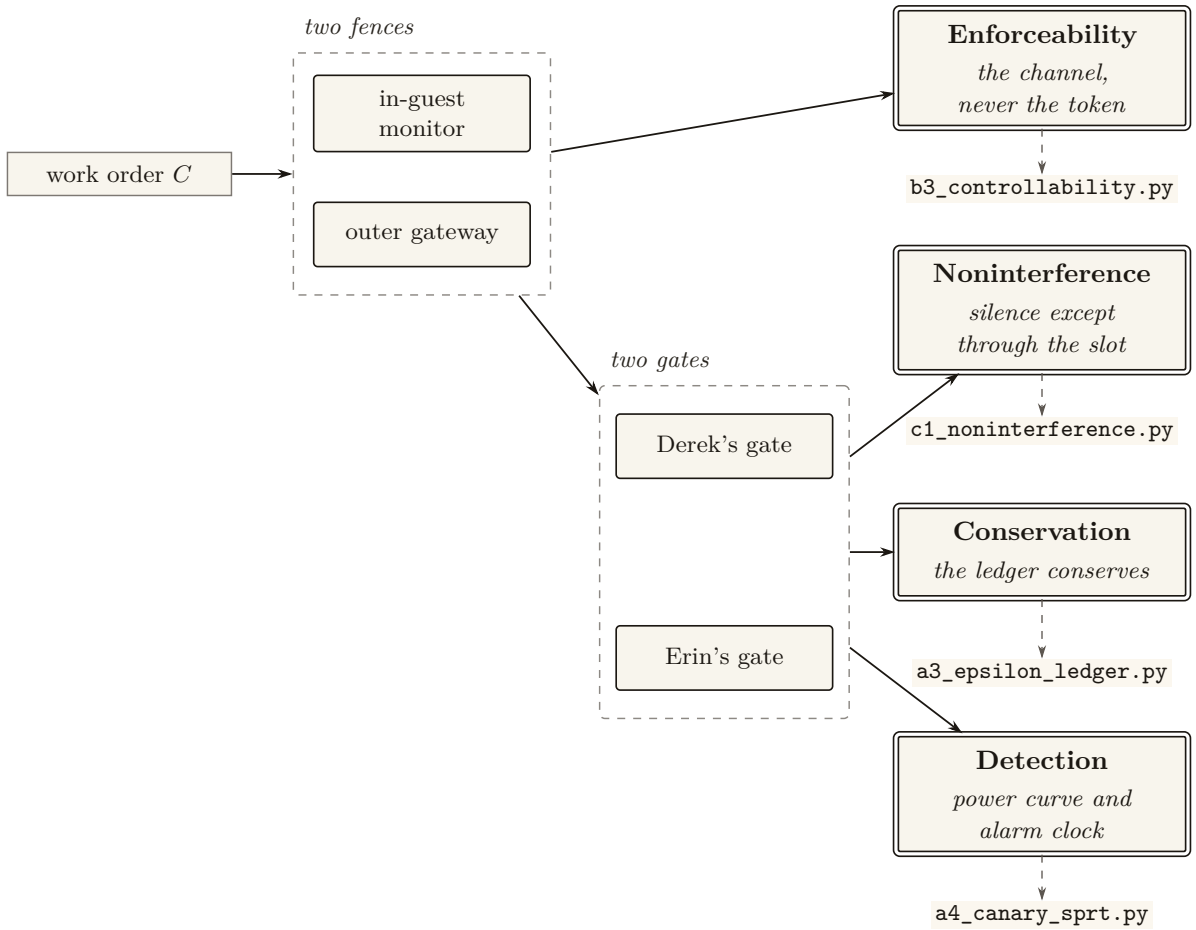


Figure 1: One work order entering the room, mediated by two fences (the in-guest monitor, the outer gateway) and released only through two gates (Derek's, Erin's); solid arrows read "produces", dashed arrows read "is checked by". The fences are what make the channel enforceable at all; the gates, once installed on that channel, are what make it silent except through the slot, what meter everything that legitimately crosses, and what are watched for anything that evades the meter – four links in one pipeline, each closing a gap the others leave open.

Recall.

1. Without looking: name the three parties who must not see each other's plaintext, and the one mechanism that seals the room from all three sides at once.
2. Why does the worker receive handles rather than bytes, even when Erin is honest?
3. Which of the six side properties is the one every later theorem assumes, and why is it a deployment property rather than a theorem?

3 Silence except through the slot

Express lane: run the whole world twice, identical except for Derek's secret, under every possible sequence of moves, and prove that Erin's view is bit-for-bit identical whenever the gate's declared function maps the two secrets to the same release; the formal statement is Theorem 3.1.

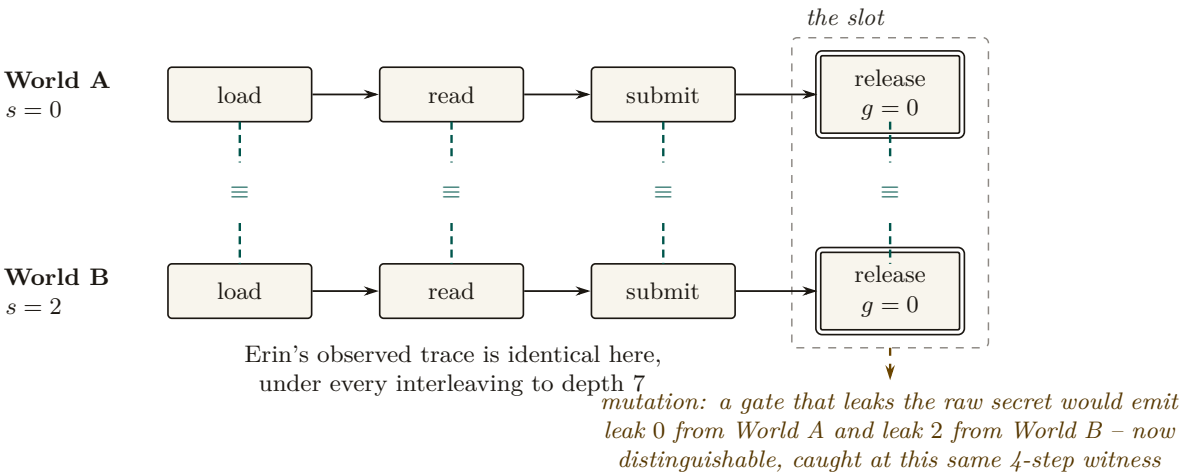
Intuition: the voting booth. You cannot inspect a wall for holes by staring at bricks; you inspect it by standing outside in two parallel worlds and asking whether anything you can see ever differs. If Derek's secret is 0 in one world and 2 in the other, and the gate is contracted to release only *parity*, 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. 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 sections price. In the booth's own vocabulary: 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.

Model-checked property 3.1 Noninterference modulo declassification on the finite clean-room model. 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. **Result.** Exhaustive over all interleavings to depth 7 and all secret pairs: zero distinguishing observations in the honest model. The schedule itself is secret-independent, checked separately, since an enabledness difference would itself be a channel. Rushby's unwinding condition of local respect, that Derek-side actions never touch Erin-observable state, holds mechanically [3]. **Mutations.** A gate that leaks the raw secret is caught distinguishing the equal-parity pair $(0, 2)$ with the witness trace `load` → `read` → `submit` → `release`, after which Erin sees `(gate,0)` in one world and `(gate,2)` in the other; a worker bypassing the gate is caught in a three-step trace. Reverting either mutation restores the property [internal, `c1_noninterference.py`].

The clause about committed input is a hypothesis, not a modelling convenience. It makes the property an instance of *delimited release* [13] 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. What the hypothesis costs is stated in the boundary below and paid in Section 7.

Proof idea. There is no proof to read; there is a search to rerun. The checker builds the two worlds side by side, one per secret, and drives them through every interleaving of the same action sequence to depth 7. At each step it projects both states onto what Erin can observe and compares. A difference at a non-release step is a counterexample; a difference at a release step is a counterexample exactly when g agreed on the two secrets. The mutation suite then shows the search has teeth: with the honest gate replaced by one that releases the raw secret, the first difference appears four steps in, on a pair the honest gate provably keeps identical. Figure 2 draws the two worlds side by side, tied step by step, with the release step boxed as the one place a difference could ever show.



Model check on the finite clean-room transition system, secrets $s \in \{0, 1, 2, 3\}$, $g(s) = s \bmod 2$: exhaustive over all interleavings to depth 7 and all secret pairs, zero distinguishing observations in the honest model. Both mutations caught: the leaked-secret gate above (4 steps), and a worker that bypasses the gate entirely (3 steps) [internal, `c1_noninterference.py`].

Figure 2: Two runs of the clean room, identical except Derek's secret, driven in lockstep through the action sequence `load`→`read`→`submit`→`release`. Erin's observable state is tied (≡) at every step; the release step is boxed as "the slot" because it is the only step the theorem lets differ at all, and here it still does not, since the two secrets share a parity. Below the slot, the leaky-gate mutation shows what a difference at that step actually looks like once caught.

Numbers by hand — six pairs, two of them equal-parity. 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 [verified]. The first mutation's witness is a four-step, human-readable proof that a particular gate breaks its contract; a security officer can read it aloud. *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? (Exercise 3.1.)

What it buys. The product's headline sentence in its provable form: *every release is explicit, gated, and bounded.* This is the finite-model core of the chapter; the general theorem over unbounded state is the unwinding obligation of Section 8, for which this model is the blueprint, the same three unwinding conditions discharged mechanically rather than exhaustively.

Where this stops. Finite depth, finite secrets: depth-7 exhaustion is a proof about the model, not the deployment. The guarantee is possibilistic; timing, token counts, and termination channels are out of model, as the design declares. The gate's *specification* is the residual oracle: the property is that releases match g, and choosing a g that leaks too much is a contract failure no theorem prevents. That risk is priced, not eliminated, by the next two sections.

There is a second residual on the other side of the same gate. The property quantifies over action sequences of a model whose gate applies g to committed input. It 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, which the delimited-release literature names and rules out by enforcement [13]. Enforcement is unavailable here, because a language model's release candidate is arbitrary output rather than a syntactically identifiable expression a checker can quantify over. The residual is therefore priced in Section 7, which is the reason the $q \cdot b$ account exists at all. The present mutation suite cannot exhibit the attack, since

> the model's `submit` carries no payload: 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 Section 8. And a human who reads a released parity may still remember it: the system bounds channels, not minds.

4 The gate sits on the only enforceable boundary

Express lane: a safety policy over an agent's behavior can be enforced by the mediation layer if and only if it constrains only events the layer mediates, so the enforceable boundary of the room is the egress channel, and any promise phrased over model-internal events is detect-only forever; the theorem is the controllability theorem of The Single-Writer Kernel*, and this section only applies it.*

The kernel chapter proved that a prefix-closed safety policy is regimentable exactly when it is controllable with respect to the uncontrollable alphabet, and classified the agent's events accordingly: the mediation layer can veto `fs_write`, `net_egress`, `exec_tool`, `git_push`, and `spawn_child`, and can at best observe `model_emit_token`, `in_context_read`, and `internal_plan`. The decisive case for this chapter is the compound rule *no `net_egress` after `in_context_read` of a secret*, and the checker's verdict is that it is regimentable: the policy permits the uncontrollable read, records it as taint, and gates only the controllable egress [internal, `b3_controllability.py`]. Gating the uncontrollable trigger, the model reading, thinking, or emitting, would be unregimentable; gating the controllable effect is not.

Table 3: Five policies over the room's alphabet, classified by the controllability theorem. A policy is regimentable exactly when the events it forbids are events the mediation layer can veto; the compound clean-room rule is regimentable because it forbids only the egress.

Policy	Constrains	Alphabet	Verdict
no `fs_write` outside the workroom	`fs_write`	controllable	regimentable
no `net_egress` except through the slot	`net_egress`	controllable	regimentable
never plan an exfiltration	`internal_plan`	uncontrollable	detect-only
never emit a token that names a secret	`model_emit_token`	uncontrollable	detect-only
no `net_egress` after `in_context_read` of a secret	**read (trigger), egress (gate)**	**mixed**	**regimentable: taint, then gate**

That verdict is the whole sealed-room design in one line, and it is the computational vindication of the correction in Section 1. Whole-worker taint is exactly what the theorem licenses: granting Hypothesis 1.1, the sound over-approximation is to taint *everything* after the first secret read and spend all enforcement on the one boundary where enforcement is possible. This is why *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 3.1.

What it buys. The two results are the same boundary proved from two independent directions. Controllability says the room *must* gate the channel and not the token, since 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 CONFINED rung of the assurance ladder, not the guarantee. The tool-level instruments of *The Single-Writer Kernel*, the guard, the shim, and the hook harness that rides inside a vendor's agent loop, sit below even that rung: they are the Observed and Coordinated modes, and the room replaces them rather than extending them, because a hook a process can decline to invoke gates nothing.

Where this stops. Controllability is relative to the mediated alphabet: partial observation shrinks the regimentable set (Lin–Wonham [5]), and policies that reference unmediated *state* need that extension, which the kernel chapter states. Detect-only is not hopeless; it is a different business, post-hoc audit and slashing, which *The Bonded Commons* treats. And complete mediation, that every effect passes through the layer, is the deployment assumption the two remaining links inherit: it is an attested property of the workroom, not a theorem.

5 The budget is a ledger, and the ledger conserves

Express lane: make the privacy budget a ledger, so that 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; the statement is Theorem 5.1 and the checked instance is Property 5.2.

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 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. The misread to preempt: conservation governs *recorded* spend. That all spend is recorded is the complete-mediation invariant of Section 2, which is why the enforceability link comes first.

Differential privacy in one paragraph. A release is ε-private when swapping any one record of D changes the probability of every possible output by at most a factor e^ε; ε is the price of the release, in a currency that adds. Two releases cost $\varepsilon_1 + \varepsilon_2$ (sequential composition); k releases of ε each cost at most $\varepsilon\sqrt{2k\ln(1/\delta')} + k\varepsilon(e^\varepsilon - 1)$ except with probability δ' (advanced composition); and a privacy filter is the bookkeeper that refuses the release which would overdraw the budget. The ledger of this section is that bookkeeper made explicit, the way double-entry accounting makes a balance explicit.

Theorem 5.1 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, because the kernel's single-writer serialization makes any concurrent execution observationally equal to some sequential interleaving. **Corollary.** The ledger is a *privacy filter* in the sense of Rogers, Roth, Ullman and Vadhan [14]: a stopping rule that halts before a pre-specified budget is exceeded. What licenses $\sigma \le \varepsilon_{\max} \Rightarrow (\varepsilon_{\max}, 0)$-differential privacy under adaptive, heterogeneous spend is their filter theorem, that summing *realised* parameters is valid, not the textbook sequential-composition theorem, which fixes the parameters in advance.

Proof. Induction over the sequential interleaving. Base: $\sigma = 0$ and $\Lambda = ()$, so the equality holds and the bound holds. Step: a refused release changes nothing; an admitted release passed the guard $\sigma + \varepsilon_i \le \varepsilon_{\max}$, so the bound is preserved, and it appends and adds in one transition, so the equality is preserved. Non-spending transitions touch neither component. Concurrency reduces to the sequential case by the linearizability of the single writer, which *The Single-Writer Kernel* proves for exactly this setting. □

Which composition bound to quote. The same authors show that substituting the advanced-composition bound into such a filter is *not* valid: the heuristic "run the advanced bound on whatever the ledger actually spent" breaks when the parameters or the stopping point are adaptively chosen. This chapter therefore quotes the advanced bound $\left(\sqrt{2k\ln(1/\delta')}\,\varepsilon + k\varepsilon(e^{\varepsilon} - 1),\ \delta'\right)$ [8] only for engagements in which k and the per-release ε are fixed in the work order, where the schedule is non-adaptive and the bound applies directly, and uses the adaptively valid filter of Whitehouse et al. [15], 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.

Whom the certificate binds. The differential-privacy reading of σ presupposes that each recorded release *is* an ε_i-private mechanism. Against the malicious worker of Section 7, one choosing b bits adversarially, that presupposition fails outright: such a worker is not running a private mechanism, and its recorded ε_i is a number it declared about itself. In that regime $\sigma \le \varepsilon_{\max}$ certifies nothing, and the $q \cdot b$ capacity account of Section 7, not the ledger, is the operative bound. The two accounts are for two different adversaries and the chapter owns both. Systems that schedule privacy budgets across concurrent jobs make the corresponding assumption explicit, that the analyst is trusted to implement the mechanism correctly [16]; this chapter declines that assumption about Erin's worker, which is precisely why it cannot lean on the ledger alone. Stating the two accounts in one currency needs the correspondence between ε and min-entropy of Alvim et al. [17], which is not discharged here.

Model-checked property 5.2 The ledger's invariants on the two-client instance. Two concurrent clients racing programs $(2, 2)$ and $(1, 2)$ against $\varepsilon_{\max} = 4$: all 15 reachable states satisfy both invariants under every interleaving, and 2000 random instances of programs, budget, and interleaving show no violation. 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.** Spend-without-append breaks conservation in one step. The torn write, which logs ε but adds $\varepsilon - 1$, breaks it in one step and in three 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. Reverting the mutations restores both invariants and the 15 states [internal, `a3_epsilon_ledger.py`].

Figure 3 grids both mutants against both invariants, with the step count each is caught in.

which mutant breaks which invariant, and how fast

mutant →	**spend-without-append**	**torn write**
ledger balance (the recorded sum)	◆ caught, 1 step	◆ caught, 1 step
budget cap (spend at or under the limit)	not exercised	◆ caught, 3 steps

Model-checked instance: two concurrent clients racing programs $(2, 2)$ *and* $(1, 2)$ *against a budget cap of* 4*. All 15 reachable states (order-sensitive; 11 as a multi-set) satisfy both invariants under every interleaving, and 2000 random instances of program, budget, and interleaving show no violation. The torn write logs the correct amount but adds one unit less: by 3 steps the recorded ledger reaches a total of* 5*, over the cap of* 4 *– an auditable over-spend. Reverting either mutation restores both invariants and the 15 states [internal,* `a3_epsilon_ledger.py`*].*

Figure 3: The two canonical atomicity breaks against the ledger's two invariants: spend-without-append (a release that adds to the balance without journaling it) and the torn write (a release that journals the wrong amount). Each caught cell marks the invariant broken and the shortest trace that catches it; the ledger's balance is the invariant both mutants break immediately, but only the torn write's deeper crime – recorded spend past the contracted maximum – takes three steps to show.

Numbers by hand — when the advanced composition bound starts to pay. At $\delta' = 10^{-6}$: for $k = 32$ releases at $\varepsilon = 0.1$, basic composition gives $32 \times 0.1 = 3.20$ while the advanced bound gives 3.31, so the advanced bound is not yet paying; at $k = 128$, $\varepsilon = 0.05$, basic gives 6.40 but advanced gives 3.30, and it pays [verified]. The crossover tells the contract writer exactly which accounting to quote at which engagement length: below it, the exact, δ-free sequential bound; above it, the $\sqrt{k}$-scaled advanced bound. *Now you try:* at $k = 8$, $\varepsilon = 0.5$, which bound is smaller, and at $\varepsilon = 0.1$, at which k does the advanced bound first pay? (Exercise 3.8.) Figure 4 plots both bounds against k and marks this exact crossing.

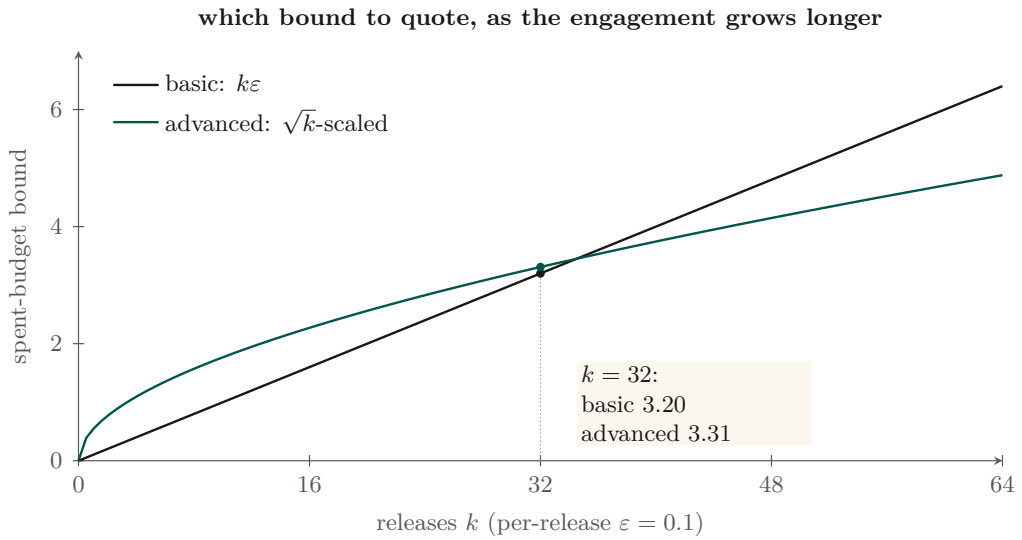


Figure 4: The basic and advanced ε-composition bounds on cumulative spend as functions of the number of releases k, at the per-release $\varepsilon = 0.1$ the chapter uses for this comparison. At $k = 32$, the chapter's reported comparison, basic composition (3.20) is still smaller than the advanced bound (3.31): below the crossover a contract should quote the exact, δ-free basic bound, and only past it the square-root-scaled advanced bound – the same rule at $\varepsilon = 0.05$ gives $k = 128$: basic 6.40 versus advanced 3.30, where the advanced bound has come to pay.

What it buys. The contract's budget line becomes an auditable invariant with the same proof shape as the bond-conservation result of *The Bonded Commons*, 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.

> **Where this stops.** Conservation governs recorded spend; that all spend is recorded is the complete-mediation invariant, so the enforceability link is the prerequisite, not a nicety. Advanced composition pays only past the crossover in k. And the theorem conserves the meter, not the meaning: choosing per-release ε_i honestly is the mechanism's obligation, outside this ledger. Differential privacy bounds individual influence in aggregates; it does not protect the whole dataset, nor Erin's model.

6 Canaries with a power curve and a clock

Express lane: 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; the statements are Theorems 6.1 and 6.2.

Intuition: dye packs and quality control. 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?" The three links before this one govern what the gate is *allowed* to pass; this one measures what happens when something tries to pass anyway. "We have canaries" is a mood; a security officer buys a number. The misread to preempt: the two theorems answer two different questions, and a single-shot exfiltration is the first theorem's job, not the second's.

Theorem 6.1 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 6.2 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 sequential probability ratio test with false-alarm target α and 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 [9, 10]. The miss target is written γ because β is already the per-canary miss rate and b is the release-channel width of Section 7.

Proof idea. Power is independence and counting: a leak is missed only if every carried canary is missed, and the number carried under uniform planting is the hypergeometric draw of m spans from n with c marked. Latency is Wald's identity: the log-likelihood ratio drifts at rate $\mathrm{KL}(p_1\|p_0)$ per output under the leak, and the test stops when it crosses the boundaries set by the two error targets, so the expected crossing time is the expected boundary value divided by the drift; overshoot past the boundary is what the identity ignores.

Verification. For Derek's corpus of $n = 10^4$ records seeded with $c = 100$ canaries (density one percent), against a channel through which each smuggled canary evades the detector with probability $\beta = 0.2$: 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 $k = 1, 2, 5$ (analytic 0.800, 0.960, 0.99968 against simulated 0.8016, 0.9608, 0.99967). The sequential test at $p_0 = 10^{-3}$, $p_1 = 10^{-2}$, $\alpha = 0.01$, $\gamma = 0.05$: Wald $\mathbb{E}_1[N] = 296.9$ against simulated 359.0, the gap being boundary overshoot, which only helps the error rates (realized miss $0.048 \le 0.05$, false alarm $0.005 \le 0.01$); $\mathbb{E}_0[N] = 431.9$ against simulated 438.1. Tenfold leak intensity cuts time to detection about thirtyfold [internal, `a4_canary_sprt.py`].

Numbers by hand — three canaries, one operating curve. At $\beta = 0.2$, two smuggled canaries are caught with probability $1 - 0.04 = 0.96$ and three with $1 - 0.008 = 0.992$ [verified]. The operating curve at $m = 100$: $1 - (1 - 0.01 \times 0.8)^{100} = 1 - 0.992^{100} = 0.553$ by the binomial approximation, against 0.554 exact [internal]. *Now you try:* at the same β, what does a five-canary leak survive with, and can you recover Wald's 296.9 from the formula with a pocket calculator? (Exercises 3.9 and 3.10.) Figure 5 plots the power curve and both expected run lengths.

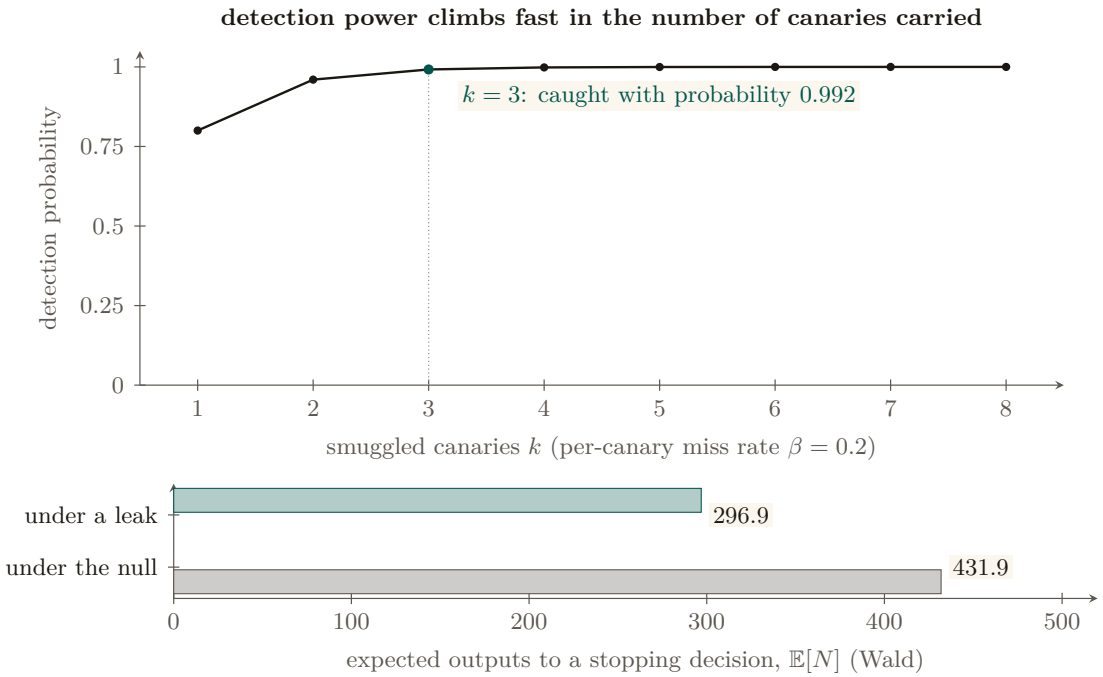


Figure 5: Top: the power curve, detection probability against canaries carried, at the measured per-canary miss rate $\beta = 0.2$; the chapter's anchor, three canaries caught with probability 0.992, is marked. Bottom: the sequential test's expected run length at a null hit-rate of 0.001, a leak hit-rate of 0.01, a false-alarm target of 0.01, and a miss target of 0.05 – Wald's formula gives 431.9 expected outputs under the no-leak null and 296.9 under exfiltration; simulation agrees up to boundary overshoot (438.1 and 359.0 respectively), which only helps the realized error rates.

What it buys. The contract can *quote* Pr(detect) as a function of leak size and expected alarm latency per leak intensity, so the exfiltration bond of *The Bonded Commons* is scoped to its real job, funding detection and response, rather than pretending to cover unbounded breach damage.

> **Where this stops.** Independence and *secrecy* of planting are load-bearing: 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 $\mathbb{E}[N]$ by about a fifth here, conservatively. And the sequential test detects *statistical* leak signatures; a single-shot exfiltration is Theorem 6.1's job.

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

The four links compose into one information-theoretic account, and the account is deliberately unromantic. It is also where the chapter pays three debts it took on earlier: the laundering residual left open by Theorem 3.1's committed-input hypothesis, the malicious worker against whom $\sigma \leq \varepsilon_{\max}$ certifies nothing, and the arbitrary-telemetry channel disclaimed in Section 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 by computing $t = f(s)$ and submitting t so that the gate honestly releases $g(t)$, can exfiltrate up to b bits through it: the noninterference property 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. In

Smith's vocabulary [7], 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 Section 5 meters, with differential-privacy composition giving the aggregate-feedback portion its formal semantics, and the canaries of Section 6 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, for 200 jobs, has granted at most 12,800 bits of adversarial capacity before timing, and can price, bond, and canary-police exactly that.

8 Limitations and boundaries

None of the following is promised, and a contract that implies any of them overclaims.

- **Zero model leakage with unlimited useful adaptive outputs to Derek.** The model-confidentiality property is weaker than the data property, structurally: Derek *intentionally observes* evaluations of M, and 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.
- **Zero data leakage with arbitrary natural-language telemetry to Erin.** That channel is the $q \cdot b$ budget; "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 by padding and 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 advice was good.
- **Literal deletion of information a party already observed**, availability if a key owner refuses or aborts, or security against a colluding hardware vendor, cloud operator, and principal; each needs a strictly stronger threat model, for which multi-party computation and homomorphic 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.

The lift. Theorem 3.1 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, and the Isabelle Archive of Formal Proofs already carries stateful intransitive-noninterference developments in exactly this lineage. The finite model is the blueprint: its local-respect check already holds mechanically, its declassification steps are the intransitive downgrader domain of Rushby's framework, and the lift consists of restating the transition relation abstractly and discharging the three lemmas once, for all depths and all secrets. One precision the lift must get

right from the start: Rushby's soundness result for these conditions is with respect to intransitive purge, but van der Meyden shows the conditions are sound *and complete* for the strictly stronger TA-security property, which additionally preserves the *order* of permitted transmissions [4]. Discharging the three lemmas therefore certifies TA-security, and the lift should state that target explicitly. Two extensions of the finite model are open alongside the lift: a `submit` that carries a payload, so the mutation suite can exhibit laundering, and a partial-observation variant of the enforceability check for policies that reference unmediated state. The status of every claim in this chapter is: the two exhaustive checks and the power-and-latency agreement are ATTESTED on their scripts; the room itself is a CONFINED design whose complete-mediation invariant is attested per deployment, not proved.

Recall.

1. Without looking: state the sentence that replaces "mathematically cannot phone home," and name the theorem that makes each of its three adjectives true.
2. Which link must hold before the conservation theorem says anything about the deployment, and why?
3. A contract grants a 32-bit status per job over 400 jobs. What is the adversarial capacity before timing, and which link polices it?

Exercises 3.1–3.12, page 17.

Review of the key ideas

What this chapter leaves coupled, in the order the rest of the book will lean on it:

1. A **sealed room** lets an agent compute on data it may not export: the data owner and the model owner each get their guarantee from a fence they can inspect, not from trust in the other.
2. **Taint does not survive a language model.** Token-level information flow through the model is not soundly definable, so the room does not try; it fences the room and gates the slot.
3. **Two fences, two gates**: the work order fixes what may enter, the slot fixes what may leave, and the model in between is treated as an opaque function of both.
4. **Noninterference is checked by running the world twice**: identical except for the secret, the two runs must agree on everything that leaves through the slot, modulo the declassification the gate performs. The finite clean-room model passes this check to depth seven.
5. The gate sits on the **only enforceable boundary**: a policy over the agent's behavior is enforceable exactly when its trigger is witnessed at the slot; anything decided inside the room is detect-only.
6. **The privacy budget is a ledger that conserves**: every release debits ε, the ledger's total never exceeds $\varepsilon_{\max}$, and a torn write is caught by the invariant within three steps on the two-client instance.
7. **Canaries have a power curve and a clock**: with per-canary miss probability β and k independent canaries, the chance a smuggled item passes is β^k (0.008 at $\beta = 0.2$, $k = 3$), and Wald's sequential test bounds how long detection takes.
8. The whole channel is priced before timing: $q \cdot b$ **bits** may leave through q queries of b bits each, and the room's guarantee is that budget, not silence.

9 Exercises

§8 Limitations and boundaries

3.1 Check ⋆ With $g(s) = s \bmod 2$ over secrets $\{0, \dots, 7\}$, how many unordered secret pairs must a sound room keep Erin-indistinguishable, and how many may differ at release steps? *(solution on page 18)*

3.2 Trace ⋆⋆ Run `c1_noninterference.py` and read the first mutation's witness. Why is the decisive pair $(0, 2)$ rather than $(0, 1)$, and at which step of the four does Erin's view first differ? *(solution on page 18)*

3.3 Check ⋆ The second mutation is caught in three steps. Which side property of Design invariant 2.1 does the mutant violate, and why is that violation detectable by the two-run check at all? *(solution on page 18)*

3.4 Open ⋆⋆⋆ Extend the finite model so that `submit` carries a payload the worker computes from the secret, and state what the checker would then have to quantify over to exhibit the laundering attack. Why is the delimited-release remedy, enforcement at the point of declassification, unavailable for a language-model worker? *(solution on page 18)*

3.5 Check ⋆ Using the controllability theorem of *The Single-Writer Kernel*, explain in two sentences why "no `net_egress` after an `in_context_read` of a secret" is regimentable but "no `in_context_read` of a secret" is not. *(solution on page 18)*

3.6 Trace ⋆⋆ Reconstruct by hand the three-step torn-write crime on the two-client instance (programs $(2, 2)$ and $(1, 2)$, $\varepsilon_{\max} = 4$): give the sequence of releases, the balance σ after each, and the recorded sum $\sum_\Lambda$, and show where the recorded sum exceeds the maximum. *(solution on page 18)*

3.7 Trace ⋆⋆ The checked instance has 15 reachable states but only 11 distinct multisets of committed releases. Explain the difference and exhibit two states that share a multiset. *(solution on page 18)*

3.8 Check ⋆ At $\delta' = 10^{-6}$: which bound is smaller at $k = 8$, $\varepsilon = 0.5$? And at $\varepsilon = 0.1$, what is the smallest k at which the advanced bound falls below basic composition? *(solution on page 19)*

3.9 Check ⋆ At $\beta = 0.2$, with what probability does a five-canary leak survive undetected? Then use the binomial approximation of the operating curve at $n = 10^4$, $c = 100$, $m = 100$ and compare with the exact value 0.554. *(solution on page 19)*

3.10 Trace ⋆⋆ Recover Wald's expected stopping time under leak, $\mathbb{E}_1[N] = 296.9$, from Theorem 6.2 at $p_0 = 10^{-3}$, $p_1 = 10^{-2}$, $\alpha = 0.01$, $\gamma = 0.05$. Why does the simulation report 359 instead? *(solution on page 19)*

3.11 Open ⋆⋆⋆ An adversary obtains the canary list with probability 0.3 and strips every canary it knows. Measured power at $k = 5$ falls from 0.9997 to 0.698. Propose one contract clause and one mechanism change that restore most of the lost power, and say what no redesign can restore. *(solution on page 19)*

3.12 Check ⋆ A work order grants Erin a 64-bit status schema per job over 200 jobs, and each job may return at one of three scheduled times. What adversarial capacity has the contract granted, before and after timing? *(solution on page 19)*

Solutions to the exercises

3.1 *(exercise on page 17)* Both parity classes have four members, so the equal-parity pairs number $2\binom{4}{2} = 12$; those must stay indistinguishable forever. The remaining $\binom{8}{2} - 12 = 16$ pairs have different parities and may differ at explicit gate-release steps only.

3.2 *(exercise on page 17)* The script prints

```
[m1 leaky gate:  releases raw s] caught:  EQUAL-CLASS VIOLATION
witness:  secrets (0,2), trace load_data -> read_secret ->
submit_to_gate -> gate_release, Erin sees (('gate', 0),) vs (('gate', 2),)
```

The pair $(0, 1)$ has different parities, so the honest gate is *allowed* to let Erin's view differ at a release step; a difference there proves nothing. The pair $(0, 2)$ has equal parity, so any difference is a violation, and it appears at the fourth step, the release, when the leaky gate emits the raw secret instead of its parity.

3.3 *(exercise on page 17)* The witness is `load_data -> read_secret -> BYPASS_write_E`, with Erin seeing `(('leak', 0),)` against `(('leak', 1),)`: complete mediation fails, since a release occurred with no gate decision. The two-run check sees it because the bypass write lands in Erin-observable state at a step that is not a gate release, which is a difference the property forbids for every pair, whatever their parity.

3.4 *(exercise on page 17)* Add a worker register t, a transition `compute`(f) that sets $t = f(s)$ for f drawn from a finite family, and let `submit` pass t to the gate, which releases $g(t)$. The property must then be stated over worlds that agree on $g(s)$ but may disagree on $g(f(s))$, and the checker must quantify over the family of f; for any non-constant f that maps an equal-parity pair to a different-parity pair, the honest gate releases a difference and the attack is exhibited. Delimited release rules the attack out by requiring that no variable used in a declassification is updated before it, which a compiler can check because the declassified expression is syntactic. A language model's release candidate is arbitrary output, not an expression the checker can identify, so the condition cannot be enforced and the residual is priced as channel capacity instead.

3.5 *(exercise on page 17)* The compound rule disables only `net_egress`, a controllable event, in the tainted state, and never disables the uncontrollable read, so its closed loop is controllable and a supervisor realises it exactly. The second rule must disable `in_context_read` itself, an event in Σ_u that the plant enables, so controllability fails on the first history that reads and the rule is detect-only.

3.6 *(exercise on page 17)* Under the torn write each release logs ε but adds $\varepsilon - 1$. Client 0 releases 2: log 2, $\sigma = 1$, $\sum_\Lambda = 2$. Client 0 releases 2 again: the guard sees $\sigma + 2 = 3 \le 4$ and admits it; log 2, $\sigma = 2$, $\sum_\Lambda = 4$. Client 1 releases 1: the guard sees $\sigma + 1 = 3 \le 4$ and admits it; log 1, $\sigma = 2$, $\sum_\Lambda = 5 > 4$. The script prints exactly this trace:

```
c0.release(2) [TORN: log 2, sigma +1] ->
c0.release(2) [TORN: log 2, sigma +1] -> c1.release(1) [TORN: log 1, sigma +0]
```

The balance never breaks the bound, so the guard keeps admitting, and the auditable log overspends.

3.7 *(exercise on page 17)* A denied release still advances the denying client's program counter, so the state records not only what was committed but which requests have already been refused. Two interleavings that commit the same releases but differ in whether a client's later request has already been refused are distinct states with the same multiset. For example, after client 0 commits both of its releases ($\sigma = 4$), the state before client 1's first request is refused and the state after it is refused carry the same ledger $\{2, 2\}$ and different counters.

3.8 *(exercise on page 17)* At $k = 8$, $\varepsilon = 0.5$: basic 4.00; advanced $\sqrt{16 \ln 10^6} \times 0.5 + 8 \times 0.5 \times (e^{0.5} - 1) = 7.43 + 2.59 = 10.03$, so quote the sequential bound, by a wide margin [verified]. At $\varepsilon = 0.1$ the advanced bound is $0.1\sqrt{2k \ln 10^6} + 0.010517k$; at $k = 34$ it is 3.42 against basic 3.40, and at $k = 35$ it is 3.48 against 3.50, so the crossover is $k = 35$ [verified].

3.9 *(exercise on page 17)* $\beta^5 = 0.00032$, so detection is 0.99968 [verified]; the script's analytic line reads `k=5:  analytic 0.99968 simulated 0.99967`. The approximation gives $1 - (1 - 0.01 \times 0.8)^{100} = 1 - 0.992^{100} = 0.553$, within a thousandth of the exact hypergeometric 0.554 [internal].

3.10 *(exercise on page 17)* $\mathrm{KL}(p_1 \| p_0) = 0.01 \ln 10 + 0.99 \ln(0.99/0.999) = 0.02303 - 0.00896 = 0.01407$. The numerator is $0.95 \ln(0.95/0.01) + 0.05 \ln(0.05/0.99) = 4.326 - 0.149 = 4.177$. The quotient is 296.9 [verified]. The simulation stops only when the likelihood ratio has crossed a boundary, and it crosses by a discrete overshoot the identity ignores; the overshoot lengthens the run by about a fifth and pushes both realized error rates below their targets, which the script reports as miss 0.0483 and false alarm 0.0052.

3.11 *(exercise on page 17)* The clause: canary secrecy is a named security boundary, so disclosure of the planting list is a breach with its own bond, and the list is regenerated per job. The mechanism: plant per job from a fresh seed held by Derek's gateway alone, so a stolen list has value for one job only and the compromise probability applies per job rather than per engagement; power then returns toward $1 - \beta^k$ for every job whose list was not taken. What no redesign restores is power against an adversary who holds *this* job's list: with the canaries stripped the leak carries none and Theorem 6.1 gives $1 - \beta^0 = 0$, which is why detection is priced as a bounded bond and not sold as a guarantee.

3.12 *(exercise on page 17)* Before timing, $q \cdot b = 200 \times 64 = 12{,}800$ bits. Three timing slots add up to $\log_2 3 \approx 1.585$ bits per job, so about 317 more bits, for roughly 13,117 bits in total; the timing term is why scheduled release with one slot is the design default.

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. Cited on page 6.

[2] J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symposium on Security and Privacy*, 1984. Cited on page 6.

[3] J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report CSL-92-02, SRI International, December 1992. Cited on pages 6 and 15.

[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. Cited on page 16.

[5] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988. Cited on page 9.

[6] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *Proc. 16th ACM Symposium on Operating Systems Principles (SOSP)*, 1997. Cited on page 3.

[7] G. Smith. On the foundations of quantitative information flow. In *Proc. FoSSaCS 2009*, LNCS 5504, pp. 288–302, 2009. Cited on page 15.

[8] 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. Cited on page 10.

[9] A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945. Cited on page 13.

[10] A. Wald and J. Wolfowitz. Optimum character of the sequential probability ratio test. *Annals of Mathematical Statistics*, 19(3):326–339, 1948. Cited on page 13.

[11] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977. Cited on page 2.

[12] 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. Cited on page 2.

[13] 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. Cited on pages 6 and 7.

[14] 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. Cited on page 9.

[15] 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. Cited on page 10.

[16] 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. Cited on page 10.

[17] 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. Cited on page 10.

[18] 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. Cited on page 1.

[19] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan. Remote ATtestation procedureS (RATS) architecture. RFC 9334, IETF, January 2023. Cited on page 3.