ATTESTED BUT UNACCOUNTED: DP-SGD PRIVACY ACCOUNTING AS A PRECONDITION OF EVERY STEP

Anonymous authors
Paper under double-blind review

ABSTRACT

Confidential training offers one attestation report as evidence for two claims: that data stays inside an attested virtual machine, and that the released model satisfies (ε, δ)-differential privacy. The report supports only the first, because it measures code, whereas ε depends on how many steps touched the data and which composition state accounted for them. We propose DP-ATTEST, which makes privacy accounting a precondition of every DP-SGD step: an attested accountant charges and seals each step against an external register before minting the token that a separate data plane requires to serve that step's batch. We measure the gap across seven accounting surfaces in six widely used packages, in real DP-SGD training and with membership audits; prove that the durable accounting state dominates the cost of every released output; and show that a fresh-once registration, which refuses every repeated sample context, makes the accountant enforce the schedule our privacy corollary assumes. No surface compares a loaded composition state with the one it holds, so DP-SGD that checkpoints, crashes and resumes touches 800 steps while accounting for 600, and at a fixed published ε, rewinding the accountant buys up to 54 accuracy points. DP-ATTEST shows no invariant violations across 235,298 enumerated interleavings of crashes, rollbacks and replays, and its gate removes the rewinding gain for a measured 0.21% of a 29.36 ms step in a CPU prototype. Binding ε to the steps a trainer ran turns a privacy claim from an assertion into an accounting any verifier can recompute.

1 INTRODUCTION

Confidential computing lets a party who does not own a machine learn what that machine is running: the hardware returns a remote attestation report naming the launch measurement of the booted image, signed under a device-rooted key and bound to a verifier's nonce (Advanced Micro Devices, 2025; Intel Corporation, 2025). Providers of confidential training now offer a bundle: the customer's data is processed inside an attested virtual machine, and the released model satisfies (ε, δ)-differential privacy (Dwork et al., 2006) for a stated ε that enters a model card (Mitchell et al., 2019) or a compliance document, with the report as its evidence. This pairing has been proposed explicitly (Duddu et al., 2025) and is already deployed (Van Overveldt & Ramage, 2025; Eichner et al., 2024).

A report measures code and configuration: it establishes which binary ran. The ε of DP-SGD (Abadi et al., 2016), by contrast, is a property of what that binary *did*: whether the batch came from the sampler the accounting assumes (P1), whether clipping was applied where the sensitivity analysis assumes (P2), whether the noise came from the accounted distribution (P3), how many steps touched the data (P4), and whether the composition state reflects exactly the steps that executed (P5). A binary that matches its measurement can still be restarted, rolled back, or cloned, and its report verifies identically. Three lines of prior work bind parts of this bundle. *Proof systems* bind the per-step preconditions P1–P3, VERIDP most completely (Abdolmaleki et al., 2026), for a *declared* step count. *State-continuity* mechanisms protect sealed state against rollback (Parno et al., 2011; Matetic et al., 2017; Angel et al., 2023); ELEPHANTDP (Jin et al., 2024) applies them to a query budget on SGX and names rollback, forking and replay; we claim no novelty for these attacks. *Budget-gated access* originates with SAGE (Lécuyer et al., 2019), which places the gate in trusted infrastructure.

The gap. None of these binds ε to the steps a trainer *ran* when the party supplying the trainer is the adversary. Continuity makes a ledger durable, but it assumes that the records reaching the ledger are

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

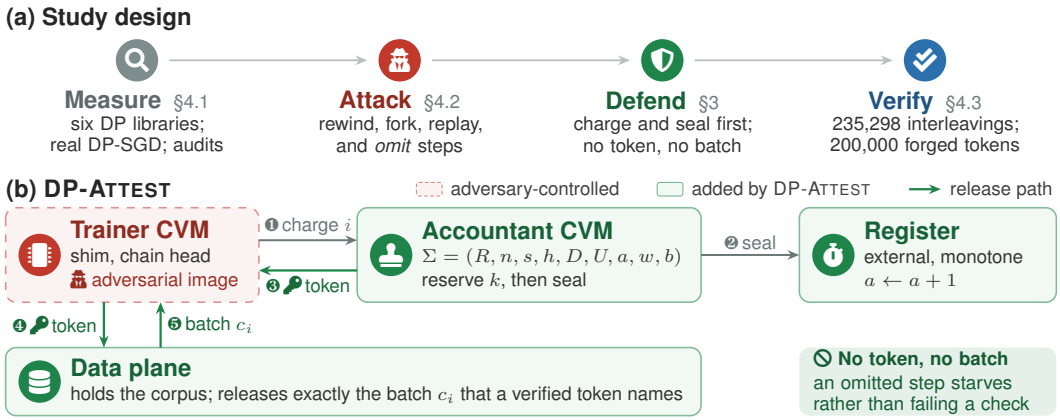


Figure 1: (a) Study design. (b) DP-ATTEST: each step is charged and sealed before its batch's token is minted.

the steps that happened. In training, unlike in a query service, the component that reads the data and the one that charges for it are separate, so an adversary inside the image need not roll the accountant back: it can simply decline to call it. Nothing observable records the missing charge: no integrity check fires, the loss curve stays smooth, and the model improves, because the extra steps are genuine training on real data.

Our approach. In this paper, we propose DP-ATTEST, which makes privacy accounting a *pre-condition* of every DP-SGD step rather than a report about it (Figure 1). An accountant runs as an attested state machine that reserves a window of budget, seals the charge against an external register, and only then mints a per-step release token; a separate data plane holds the corpus and serves step i's batch only against that token. An adversary that omits a step does not fail a check: it starves. Our contributions are:

- **A measurement of the gap.** None of the seven accounting surfaces across six widely used packages compares a loaded composition state with the one it holds; DP-SGD that checkpoints, crashes and resumes touches 800 steps while accounting for 600; and a membership audit detects the iterate a crash discards at better than chance (Section 4.1).
- **The omission attack.** A trainer that runs 41,098 steps and reports 27,399 publishes $\varepsilon = 1.00$ where the executed trace recomputes to 1.17, and no continuity mechanism detects it (Section 4.2). Unaccounted steps are worth having: obtained by rewinding the accountant, they buy 14.1 accuracy points at $\varepsilon = 1.0$ and 54 at $\varepsilon = 0.5$ for an unchanged published ε (Section 4.3).
- **DP-ATTEST**, a write-ahead accountant and a gated data plane whose bookkeeping invariant we prove for every interleaving; under a fresh-once registration it also enforces the schedule our privacy corollary assumes (Section 3).
- **An evaluation of the defence.** Accounted steps never fall below executed steps under any crash schedule, with no violations across 235,298 enumerated interleavings; under fresh-once, no context carries a second output across 118,098 further interleavings; 0 of 200,000 forged tokens are served; and the gate costs a measured 0.21% of a step in a CPU prototype (Section 4.3).

The rest of this paper is organized as follows. Section 2 states what ε depends on and the threat model; Section 3 presents DP-ATTEST and its guarantee; Section 4 measures the gap and evaluates the defence; Section 5 interprets the results, relates them to prior work and discusses threats to validity; Section 6 concludes.

2 BACKGROUND AND THREAT MODEL

What an attestation report covers. A confidential VM's launch measurement, carried in an AMD SEV-SNP report (Advanced Micro Devices, 2025) or in Intel TDX's measurement register (Intel Corporation, 2025), covers code and configuration, not data or results, and is fresh with respect to a nonce rather than to a history: it attests that an instance of the measured image is running now, not how many instances or launches there have been or from what state one resumed. Sealed storage provides integrity at rest but not freshness, so an adversary that controls the platform can present a stale sealed blob, which the enclave accepts as authentic; freshness requires an external register (Parno et al., 2011; Strackx et al., 2014; Matetic et al., 2017; Angel et al., 2023).

What ε depends on. Throughout, (ε, δ) is example-level under add/remove adjacency, and the privacy adversary is anyone who receives the released model or observes storage; a published dataset commitment must therefore be hiding, as ours is (Section B.1). DP-SGD (Abadi et al., 2016) samples a batch, clips each per-example gradient to norm C, sums, adds Gaussian noise of scale σC, and updates the model. The accountant never sees a gradient: it is a state machine holding $\Sigma = (R, n)$, where R accumulates the Rényi divergence (Mironov, 2017) at each order and n counts accounted steps; a step at rate q and noise multiplier σ adds $\rho(q, \sigma)$ to R and one to n, and the reported guarantee follows by conversion at δ (Mironov et al., 2019; Balle et al., 2020). Composition is commutative, so Σ keeps no record of which steps contributed, in what order, or how many times.

Three quantities, never interchangeable. **Reported** ε is what an accountant published for the trace it was told about. **Recomputed** ε is the *same* accountant's output for the trace that actually executed — an upper bound, not the minimal privacy parameter of the release. **Audited** ε is an empirical *lower* bound from a one-run membership procedure (Steinke et al., 2023). A recomputed value above a reported one shows that the reported accounting does not certify the executed trace, which we call *unsupported accounting provenance*; it does not show that the release violates the reported (ε, δ).

Five preconditions. The reported ε is valid only if **(P1)** the batch came from the sampler whose amplification the accounting claims — Poisson subsampling at rate q earns a credit that a shuffling loader does not obviously earn (Chua et al., 2024); **(P2)** the clip was applied to the quantity whose sensitivity the noise is calibrated against (Balle & Wang, 2018); **(P3)** the noise came from the accounted distribution, with randomness the adversary could neither predict nor grind; **(P4)** accounted steps equal the number of times the data was touched; and **(P5)** the accumulator reflects exactly the multiset of steps that executed. P1–P3 are per-step and stateless, which is why verifiable training enforces them in a per-iteration circuit; P4 and P5 are properties of an execution, held in mutable state that outlives any measured process, and no per-step proof establishes either (Figure 4). The composed ε grows roughly with the square root of the executed count, so inflating the count is a sublinear, quiet attack.

Released outputs, and the accountant as a service. DP-SGD's analysis charges iterates that a preemption discards; we test, rather than assume, that they are observable (Section 4.1). *Snapshot*, *restore* and *fork*, ordinary operational events that no privacy analysis models, place Σ in storage the platform administers (Section A.1). The accountant is in no party's trusted computing base: it is not the measured trainer, and its output is the very quantity being asserted.

Threat model. A customer or regulator wants assurance about a model trained on sensitive data by a party it does not control, in the form of an attestation report over the trainer and a claimed (ε, δ). We assume the pipeline is correct in every respect that current practice can check, and ask what remains unconstrained.

Trust assumptions. We trust the CPU and GPU TEEs, their firmware and key hierarchies, the vendor attestation services, the cryptography, the relying party's checks, and that the trainer binary is the one the launch measurement names. Everything else is untrusted: the contents and order of the data stream, the persistent state of any service outside the measured image, and how many times that image has been launched. *No attack here needs a trusted component to fail.* Three assumptions are ours. DP-ATTEST needs a register, outside the accountant's rewindable persistence, that an adversary

can neither lower, stall, nor advance on another holder's behalf; no confidential VM provides one (Section C.5), and Section 4.3 attacks ours. It also needs the sampler's randomness to remain secret from the party against whom the guarantee is claimed; Table 30 prices a violation. Last, the host cannot clone a *live* accountant–plane pair inside a prepaid window (condition (vii) of Proposition 1).

Attacker. The primary adversary is a *compromised component* inside an otherwise honest deployment (Section A.1): a compromised dependency of the training job (data loader, sampler, or DP library) that runs inside the measured image while the measurement stays exactly as the customer pinned it; a data pipeline outside that image; or the accountant service, whose persistence it can rewind or fork without touching the trainer. Its goal is to weaken the guarantee while the reported number is unchanged and every verification step succeeds, with no failed run, divergent loss curve or suspicious log. Whether it *retains* the data it is given decides what any data-gating defence can promise: **A1** manipulates control flow but holds a batch only for the step that uses it, while **A2** also keeps every record it is served. Against A1 a gate on the data is a complete answer; against A2 it bounds *fresh* access, not memory, which is why our guarantee is stated over A2 and the gradient plane (Section 3.2). Some losses need *no adversary at all* — a crash loop on preemptible capacity rewinds an accountant checkpointed with the trainer — and we include them: a deployment that cannot tell a crash loop from an attack cannot make the attack accountable.

3 DP-ATTEST

DP-ATTEST binds the privacy-relevant preconditions of a training run rather than attesting the run itself, which would place the whole training stack inside a measurement nobody can review and would still leave ε unbound. It attests a small state machine that consumes committed records and refuses to report a number those records do not support (Figure 1b).

A certificate of (ε, δ) needs four properties that no attestation of a trainer binary provides. *Completeness*: every step that touched the data is represented by exactly one accounted record, duplicates included. *Monotonicity*: accountant state never regresses across crashes, preemptions and restarts. *Join*: a parallel run is certified only against a state that has consumed every branch. *Faithfulness*: each record's mechanism parameters are the ones the step used. DP-ATTEST enforces the first three, which are systems properties of a stateful service, but not the fourth: (q_i, σ_i, C_i) are pinned at registration, yet a compromised noise path yields a certificate that is correct for the parameters the adversary declared; VERIDP binds these in zero knowledge. Completeness dictates the design: a record that is never emitted cannot be examined, so the omission attack (Section 4.2) defeats every accountant that is a passive recipient, and completeness must be enforced on the *data*.

Neither the step records nor the sampling and noise are ours: a shim inside the trainer's CVM hash-chains records per replica, and VERIDP binds sampling and noise. The batch for step i must, however, come from a sealed seed: against an adversary who reconstructs the sample, only the unamplified Gaussian's guarantee holds, and its ε is $317\times$ larger over 200 steps (Table 30 and Section A.2).

3.1 THE ACCOUNTANT AS AN ATTESTED STATE MACHINE

The accountant runs in its own confidential VM, whose measurement covers exactly the transition function and the budget policy. Its state extends the (R, n) of Section 2 with the fields whose absence the attacks exploit, $\Sigma = (R, n, s, h, D, U, a, w, b)$: the surcharge units s, which with n pin R; the chain head h per replica; the record D of committed indices and the set U of upgraded sample contexts; the anchor value a; the units w of the current reservation charged but unspent; and the per-replica index bound b that the sealed window could release. Σ is sealed to the accountant's measurement, with a inside the blob.

Monotonicity. COMMIT advances the *anchor* — the external register of Section 2 — by compare-and-swap from the generation the caller believes it holds, checks that it advanced, and only then seals Σ; AUTHORISE reserves a window, commits it, and mints a token against it (Algorithm 1; RESTART is Algorithm 2). Increment-then-persist leaves a crash inside the window with sealed state *behind* the anchor, which fails closed, whereas persist-then-increment makes a genuine rollback indistinguishable from an ordinary crash. The advance is *conditional*, so the anchor records *whose*

state advanced; with a bare increment, two clones of one sealed blob each seal a divergent history and serve 60 steps unnoticed.

Write-ahead reservation. Charging in memory and sealing every k steps is not enough, and the attack that breaks it is not a rollback: a plain crash leaves the sealed counter *equal* to the anchor, so the restart looks clean, although the charges since the last seal are lost and their batches have already been served. DP-ATTEST therefore writes the charge *ahead*: a window of k steps is added to Σ and sealed *before* its first token is minted. A token is minted only by decrementing w; w rises only through a reservation that charged n and sealed first; and recovery resets w to zero and advances D through the window bound b. A sample first used after the last seal therefore cannot return after a restore looking fresh — its sample context is skipped, or recognised as a repeat and priced unamplified — so each mint maps to a distinct durable unit, and the accounted step count A stays at or above the executed count E at every interleaving. The price is an unused tail of at most k steps charged on a crash; the gate stops a run only when a *new* reservation is needed (Algorithm 1, line 11), so a defended run spends exactly its budget.

Recovery from an older seal. The adversary may present an older sealed blob, not only the latest, and bounding what the lost generations did is unsound, because recovery raises $D.\mathrm{high}$ and the next reservation extends the bound from the raised value. The register instead carries the *accounting frontier* $F = (\max_j b_j,\, n,\, s)$, three nondecreasing integers, and recovery *reconstructs* the state from it exactly, since $R = n\rho(q,\sigma) + s\Delta$; with no frontier to read it refuses rather than guesses (Section A.2).

Completeness: gate the data, then charge. Completeness is enforced by a third party, the *data plane*, the only component that can release a batch; it holds the capability to decrypt rather than the bytes, so object storage, sharding and prefetching remain possible (Section C.2). To execute step i the trainer asks the accountant to charge it; the accountant advances Σ and only then returns a release token — a MAC, under a key shared with the plane alone, over (rid, j, i, c_i, u_i, a): the batch commitment the accountant derived, the query, and the generation that charged it. A re-executed step is charged again and served again; a verbatim replay gets the bytes it already received. Forks fail at the gate too: the accountant advances only on a record whose predecessor hash matches the head it holds for that replica, and replicas are declared at registration (Section C.3), so a silently spawned replica's records are refused.

3.2 THE GUARANTEE, AND THE DEPLOYMENT THAT CARRIES IT

The two planes do not support the same claim, so we state it for one. **The primary configuration is the gradient-releasing plane against A2**: the plane holds the records and the mechanism, only a noised gradient crosses its boundary, and the record plane is reported as the weaker variant it is (Table 32). *The adversary sees* every object the plane releases, one per redeemed capability, plus the capabilities' fields, the certificate, the anchor values and commitments to the sampler's seed, which no party can grind and the adversary cannot reconstruct (Sections A.2 and B.1); iterates, checkpoints and the released model are functions of that transcript. Write A for the accounted step count, R for the Rényi vector that would survive a crash at this instant, and M for the number of *distinct* mechanism outputs released.

Proposition 1 (the durable vector dominates the released transcript)**.** *In the primary configuration, at every reachable state of every interleaving of authorisations, releases, crashes, restores and* repeated redemptions*: $A \geq M$, and coordinatewise at every Rényi order $R \succeq \sum_{o\in\mathcal{O}} \rho(o)$ for $\mathcal{O}$ the distinct released objects and $\rho(o)$ the strict price of o, set by how many outputs share its sample context — provided that:*

(i) the register is monotone and per tenant in the accounting frontier $F = (\max_j b_j,\, n,\, s)$ *as well as in the generation, and carries the lineage digest;*

(ii) the plane solely holds the plaintext records and the mechanism;

(iii) every capability names inside its MAC the query u_i it was charged at;

Table 1: What a certificate establishes, layer by layer.

Layer	Statement	Premises, and who establishes them
Bookkeeping	Proposition 1: $A \geq M$, $R \succeq$ released cost	(i)–(vii); the verifier recomputes counts, heads, anchor, ε
Ideal privacy	Corollaries 1 and 2: Rényi bound	(F), (P); fixed, non-interleaved schedule: **enforced** fresh-once, else a deployment obligation
Deployed	computational $(\varepsilon, \delta + \mathrm{negl})$	($S^\star$), PRF, PRG: assumed
As run	bookkeeping, CPU prototype	both planes; PCG64 noise; no confidential hardware

(iv) the plane's mechanism randomness is a pseudorandom function of the capability under a seed sealed to its measurement;

(v) any cost of a release above the unit that prepaid it is sealed before that release's capability is minted, and the mechanism parameters are the declared ones, so that $R = n\rho(q, \sigma) + s\Delta$*;*

(vi) fresh step indices are contiguous per replica, and recovery from a state the register does not name restores F *and otherwise refuses (Algorithm 2); and*

*(vii) every instance start passes registration/*RESTART*, while the deployment prevents live RAM cloning of an active accountant–plane pair within a reserved window (or sets* $k = 1$*).*

*This is a **bookkeeping** statement; the **privacy** corollary (Corollary 1) holds for an ideal experiment and needs a schedule fixed before the run and* non-interleaved *(Table 1).*

Conditions (iii) and (iv) are not free: without them one charged window bought 41 distinct noised gradients in *our own* implementation (Section A.4). Section B.1 proves the proposition; Section 4.3 checks it against the implementation.

What the certificate certifies. Two things, and the verifier tells them apart (Table 1). It always certifies *bookkeeping*: counts, chain heads, the anchor and an ε it recomputes for the trace that ran. It certifies the privacy corollary only when the accountant was registered **fresh-once**, refusing every authorisation on a sample context it has already authorised, a test that stays sound across recovery. Each context then carries at most one output, so every release order is non-interleaved; and the schedule is the one committed at registration, at most $T^\star$ fresh contexts at the pinned (q, σ), so a run that adaptively skips or stops early is a post-processing of it and the certified value is $\varepsilon(T^\star) \leq \varepsilon_{\max}$ (Corollary 2). With repeats priced instead, both premises are deployment obligations that counts cannot show, so interleaved re-executions are covered by Proposition 1 but not by the corollary. On the record plane a charge buys a batch release, not a mechanism invocation: *access* accounting. Everything here is built, the register included (Sections A.2 and C.1).

4 EVALUATION

The analytic regime was fixed before any attack was run: $q = 0.001$, $\sigma = 1.0$, $\delta = 10^{-5}$, a budget of $\varepsilon = 1.0$, and a checkpoint every 500 steps; an honest run exhausts the budget after 27,399 steps. Table 20 labels the kind of evidence behind each result.

4.1 DO DEPLOYED ACCOUNTANTS ACCEPT A REWIND?

Of the seven accounting surfaces across six packages (Table 5), exactly one, Opacus, ships a checkpoint path that carries composition state across a restart; the others compose over a step count supplied by the caller. Our claim is narrow: none compares a loaded state with the one it already holds, including the one that carries such state. Driving Opacus 1.6.0 to a reported ε of 4.08 and reloading an earlier checkpoint returns the report to 2.87 without warning; repeated for 8 cycles, the loop executes 14,068 steps against 500 accounted while still reporting 1.653 (Table 17). The ordinary failure needs no adversary (Table 25): in real DP-SGD that checkpoints, crashes and resumes, `load_checkpoint` restores the accountant's history without comparing it to the one it holds, so the steps between checkpoint and crash are never charged: 800 executed, 600 accounted.

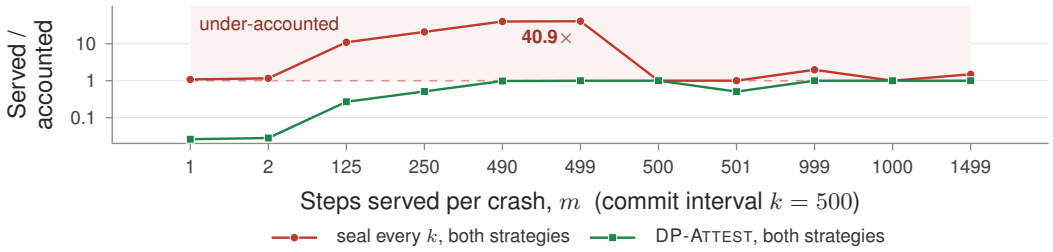


Figure 2: Steps served per step accounted, across crash schedules.

Are the discarded steps owed, and do they leak? Charging discarded steps is justified only if a discarded iterate is observable, so we audit one over 12 seeds (Table 23): its audited bound is 0.116 $[0.086, 0.145]$, against 0.118 for the published model, both separated from chance at $p < 2\times10^{-49}$. Unaccounted steps also leak measurably more: at an identical reported ε of 9.7, a one-run membership bound (Steinke et al., 2023) rises from 0.071 to 0.157 $[0.141, 0.172]$ with the executed count over 15 seeds (Figure 6), a trend rather than a certified violation. At deployment noise ($\sigma = 1.5$) this audit family certifies no more than 0.26 (Section A.4), so the remedy must be prevention.

4.2 THE ADVERSARY INSIDE THE IMAGE

The adversary built the image, so its cheapest attack is not to roll the accountant back but simply not to inform it: it exploits the one assumption every state-continuity mechanism makes and Section 2 does not grant — that the accountant is told of every step that ran. A trainer that reports 27,399 of the 41,098 steps it runs publishes $\varepsilon = 1.00$ where the executed trace recomputes to 1.17, and 0 of the 13,699 hidden steps are detected by any continuity mechanism, ours included, because nothing contradicts a record that was never written. Making accounting a precondition removes the choice: without a token the plane serves nothing, and the batch it serves is the one the sampler determines. All 13,699 hidden steps are refused; beyond the 4 evasion routes we designed, we fuzzed the gate with 200,000 randomised tokens across 8 mutation classes, of which 0 were served. This settles A1. Against A2 the exposure is the run rather than a buffer: coverage saturates at 95% of the corpus after 3,000 compliant steps, out of 27,399 in an honest run (Table 22), so A2 can comply for a prefix, accumulate data, then train off the books; hence Proposition 1 is stated for the gradient plane, where no record crosses the boundary.

4.3 WHAT THE DEFENCE STOPS, AND WHAT IT COSTS

Table 4 re-runs every attack against DP-ATTEST: forking by an undeclared replica is prevented, whereas replay is *not*, and should not be, because the re-executed step touched the data; each recomputed column refers to its own trace (Section A.4).

Crash schedules. Sealing every k steps and write-ahead reservation both ship in the artifact, and Figure 2 sweeps them over 22 crash schedules and both crash strategies. Sealing every k fails as a sawtooth rather than as a constant tax: a crash just before a seal yields 40.92 served steps per accounted step. Write-ahead never exceeds 1.00 under any schedule, and adding a rollback to the crash changes neither curve: the window was sealed before it ran, and the frontier already records what the crash lost (Table 34).

Exhaustive interleavings. A proof about a protocol is not a statement about its implementation, so we enumerated every ordering of *authorise*, *serve*, *replay*, *crash* and *roll back* to depth 6 on both planes: 235,298 sequences, with no violations of $A \geq E$, $A \geq M$, or the coordinatewise dominance of Proposition 1. *Re-authorise* and *roll back deep* reach surcharge-only commits and older blobs (118,098 further sequences at depth 5, no violations). The checker's oracle keeps every cost outside the accountant and prices what the plane released, per sample context. Under *fresh-once* registration, the same alphabet at depth 5 on both planes (118,098 sequences) never mints a second capability or releases a second output on any context, and every release order is non-interleaved (no violations);

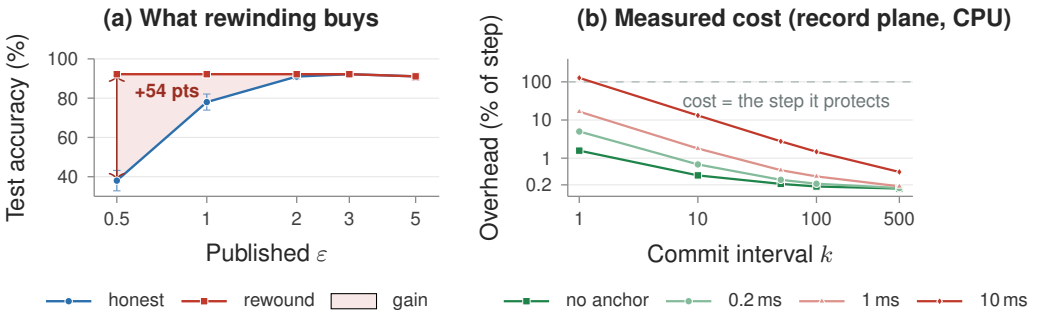


Figure 3: (a) Accuracy at a fixed published ε. (b) Overhead by commit interval k and anchor latency.

the priced accountant, run under the same check as a control, violates it in 387 of 6,561 sequences at depth 4. Targeted regressions cover what no tractable depth reaches (Sections A.2 and A.3).

The register under attack. Everything rests on a register the adversary cannot rewind, and Table 24 attacks it at 17 positions: a rolled-back store is detected and recharged, a rewound register is refused, a lost quorum fails closed, a split brain is detected afterwards, and a replayed *authentic* reply is refused.

What the gap is worth. Table 6 prices the gap in the quantity an operator optimises: DP-SGD on UCI optical digits with the gate in the loop, at $\varepsilon = 1.0$ over 40 seeds. An honest run spends its budget in 65 steps for 77.8% accuracy; a run that rewinds only its accountant publishes the *same* ε, executes 585 steps that recompute to 3.01, and reaches 91.9%: 14.1 points gained while the published number stays fixed. Under DP-ATTEST the gain disappears (77.8%), and the gradient plane, which also removes the retention residual, reaches 77.9%. A predeclared paired equivalence test over 40 seeds and a ± 2-point margin supports equivalence of the record plane to honest training and supports equivalence of the gradient plane to the record plane (Section A.4). Figure 3(a) sweeps the budget: the gain grows as the published ε shrinks, to 54 points at $\varepsilon = 0.5$, and vanishes by $\varepsilon = 3$, where the honest budget already buys a converged model.

What the defence costs. Figure 3(b) is the *measured* end-to-end cost against a 29.36 ms DP-SGD step, with a real, authenticated quorum register on a socket and the plane in the loop. Prefetched, DP-ATTEST adds 62.7 µs per step at $k = 100$ and a 0.2 ms anchor, or 0.21%; at $k = 1$ a 10 ms anchor costs 127%, so a register far from the trainer calls for a larger k. *Calculated* against measured step times from 3.3 ms to 1,501 ms, the same added cost is 1.9% to 0.004% of a step (Table 28). The window is also a utility price: at a 10.0 ms round trip, the *calculated* anchor time is 10.15 ms per step at $k = 1$, with 73.1% accuracy, and 1.32 ms at $k = 20$, with 60.3% on 38 useful steps against 27.0 forfeited (Table 8); both settings are safe. These costs are for the record plane on a CPU; the gradient plane adds a measured 25 µs (Table 29), and a confidential-GPU figure is a *projection* (Section C.2).

5 DISCUSSION

What the findings mean. The measurements share a single cause. An attestation report is a claim about a *state*, evaluated once at launch; a privacy guarantee is a claim about a *history*. Every gap we measured comes from asking the first to certify the second, which is also why the losses are invisible: each attack has an operational excuse — rollback is crash recovery, forking is data parallelism, replay is preemption — so there is no event to detect, and Table 25 finds the same loss with no adversary at all. Making accounting a precondition turns that history back into checkable state: a durable, externally anchored step count that the data cannot run ahead of.

Relation to prior work. Work on trustworthy DP training divides by what it assumes of the implementer (Table 13). Auditing and attack work (Cebere et al., 2026; Chua et al., 2024; Mironov, 2012; Jin et al., 2022; Sun & He, 2026) studies implementers that are careless or evade detection; our adversary passes every verification. VERIDP (Abdolmaleki et al., 2026) binds strictly more of

P1–P3 than we do (batch membership, resistance to seed grinding, clipping, in-circuit noise and the composed Rényi curve), but for an iteration count that is a public parameter of the proved statement: it certifies the arithmetic for a *declared* count, not that the state survived a restart, fork or replay. The two would compose; we have not built that composition. ELEPHANTDP (Jin et al., 2024) identifies rollback, forking and replay against a privacy budget in a TEE, but in a query service the TEE that answers *is* the TEE that charges, whereas in training the component that reads the data and the accountant are separate, which is what the omission attack exploits. Budget-gated access (Lécuyer et al., 2019; Luo et al., 2021; Tholoniat et al., 2025) places its gate in trusted infrastructure; moving the gate under an untrusted host is what necessitates the register, the token and the write-ahead window (Section A.5).

Unexpected and negative results. Three results ran against our expectations. First, the natural design, which seals every k steps, passes every rollback test in the literature yet fails under plain crashes (Figure 2): continuity protects the wrong quantity. Second, checking the proof against the implementation exposed defects in natural variants of our protocol, including one where a single charged window bought 41 noised gradients (Section 3.2); each compared the wrong pair of values in a freshness test (Section A.3). Third, the audits we expected to corroborate the attacks have no statistical power at deployment noise levels (Section 4.1), so a relying party cannot detect an unaccounted run by auditing the model.

Implications. For *researchers*, a published ε should come with a transcript a third party can recompute, the sampler it assumes, and whether it is access or mechanism accounting. For *practitioners*, the cheapest binding is also the most valuable: one external register and a conditional advance make rollback and replay visible. For *tool builders*, DP libraries should compare a loaded composition state with the one they hold and bind it to a run and dataset identity, and platforms should expose the monotone register none provides today (Section C.5).

Threats to validity. *Construct*: our recomputed columns are the same accountant's upper bound for the trace that ran, so they show unsupported provenance rather than a demonstrated violation; the privacy corollary covers an ideal experiment with a schedule fixed in advance and non-interleaved in release order, which the fresh-once registration enforces and a verifier cannot otherwise check (Table 1); and nothing ran on confidential hardware, so Figure 3(b) is a CPU prototype's number. *Internal*: the library study reports an absent check in components whose threat model never included a rewinding caller, and shows neither a defective library nor an exposed deployment; the audits are lower bounds, none of which crosses a published budget. *External*: the analytic results use one (q, σ, δ) and the training one small dataset. A (q, σ) grid (Table 12) shows the invariant is parameter-free, the attacks' recomputed ε stays within 1.17–1.46 wherever the budget buys at least 50 steps, and what ten crashes forfeit depends on that step count: 69% at $k = 10$ for the training run's 65, 1.8% at $k = 100$ analytically. *Conclusion*: accuracy comparisons rest on 40 paired seeds with a predeclared ± 2-point margin and the audits on 15 seeds, so "equivalent" means bounded, not zero (Section A.5).

6 CONCLUSION

An attestation report tells a relying party which binary ran; a differential-privacy guarantee depends on what that binary did to the data, and no existing mechanism binds the two when the trainer's supplier is the adversary. DP-ATTEST ties them by charging every DP-SGD step in an attested, externally anchored accountant before the data plane serves its batch. Released accounting software accepts rewound state, ordinary crash-and-resume training under-accounts, and a rewound accountant buys up to 54 accuracy points at an unchanged published ε; DP-ATTEST removes that gain at a measured 0.21% of a step, with no violations across 235,298 enumerated interleavings. A published ε thereby becomes an accounting bound to the steps a trainer ran, which a verifier recomputes rather than trusts and, under fresh-once registration, whose privacy corollary rests on premises the accountant enforces. The guarantee covers the gradient-releasing plane, the costs come from a CPU prototype, and interleaved repeats of a context lie outside the corollary. Future work will run DP-ATTEST on confidential GPUs with a hardware register, compose it with a proof system such as VERIDP to bind mechanism parameters, and extend the corollary to adaptive, interleaved schedules.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods and refactor the evaluation and implementation code; to design experiments and regression tests, including the adversarial schedules against the register and the sample-context traces; to help develop the accounting framework, formulate and refine mathematical claims and hypotheses, and assist in writing their proofs (Section B); and to interpret results. We have not used generative AI tools to generate data sets — the synthetic corpora come from a seeded program in the artifact, and the real one is the public UCI optical-digits set, prepared only by a fixed pixel scaling in the released code — and qualitative or thematic data analysis is not applicable to this work. Additionally, we used generative AI tools to draft, restructure and copy-edit the paper, draw the TikZ figures, analyse feedback on drafts, format references and propose the title. We have reviewed all AI-assisted work: no reported number is a language-model output, since every number in the tables and prose is generated from the retained experiment records, which the released code reproduces, and every AI-assisted proof, claim and code change was checked against the tests, the stored results and the cited sources. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This paper reports an absent check in released, widely used software and builds an attack that a provider could use to overstate a privacy guarantee, so we state the balance explicitly rather than leaving it implied; Section D.1 gives the full treatment. No third-party system was attacked: every experiment runs on infrastructure we control, and the seven accounting surfaces of Table 5 were exercised locally through their own public APIs, with no hosted service probed and no other user's data touched. No personal or human-subjects data is involved — the training corpora are synthetic or the public UCI optical-digits set, and the canaries are our own constructions. The paper names versions rather than attributing fault, since a library that never promised rollback resistance is not defective for lacking it. Section D.1 describes how we approached disclosure. On dual use we publish, because the capability is not created by this paper — the checkpoint API is documented and the missing check is visible in a few lines of published source — because the party who benefits from secrecy is the one misreporting rather than the data subject, and because the defence travels with the attack in the same artifact.

REPRODUCIBILITY STATEMENT

Every number in this paper can be reproduced on a CPU, and no result depends on hardware, data or software that a reader cannot obtain. The anonymous supplementary archive contains the prototype, every attack and experiment, the test suite and the recorded result records with their provenance, and its README gives one command per result family; Section D.2 maps each table and figure to the record and script behind it (Table 36), and Section C.1 describes the implementation and its tests. The paper's tables and quoted numbers are generated from those records rather than typed. The analytic regime (q, σ, δ, budget, checkpoint interval) is fixed at the head of Section 4; Section B gives the proofs and the closed forms each analytic number is checked against; and Table 20 labels every result by the kind of evidence it is. Nothing here was run on confidential hardware, which Section 5 treats as a threat to construct validity.

REFERENCES

Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, pp. 308–318. ACM, 2016. doi: 10.1145/2976749.2978318.

Behzad Abdolmaleki, Amir R. Asadi, Vahid R. Asadi, Stefan Köpsell, Bhavish Mohee, Nahid Roustaeifar, and Maryam Zarezadeh. VeriDP: Verifiable differentially private training. *Proceedings on Privacy Enhancing Technologies*, 2026(3):48–64, 2026. URL `https://petsymposium.org/popets/2026/popets-2026-0070.php`. Also Cryptology ePrint Archive, Paper 2026/542.

John M. Abowd, Robert Ashmead, Ryan Cumings-Menon, Simson Garfinkel, Micah Heineck, Christine Heiss, Robert Johns, Daniel Kifer, Philip Leclerc, Ashwin Machanavajjhala, Brett Moran, William Sexton, Matthew Spence, and Pavel Zhuravlev. The 2020 census disclosure avoidance system TopDown algorithm. *Harvard Data Science Review*, (Special Issue 2), 2022. doi: 10.1162/99608f92.529e3cb9.

Advanced Micro Devices. SEV secure nested paging firmware ABI specification. AMD Publication #56860, 2025. URL https://www.amd.com/content/dam/amd/en/documents/developer/56860.pdf. Retrieved 2026-08-06.

Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath Setty, and Sudheesh Singanamalla. Nimble: Rollback protection for confidential cloud services. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI '23)*, pp. 193–208. USENIX Association, 2023. URL https://www.usenix.org/conference/osdi23/presentation/angel.

Meenatchi Sundaram Muthu Selva Annamalai and Emiliano De Cristofaro. Nearly tight black-box auditing of differentially private machine learning. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/ed93b2b5722acc2341d421b8916404a1-Abstract-Conference.html.

Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. SPEICHER: Securing LSM-based key-value stores using shielded execution. In *17th USENIX Conference on File and Storage Technologies (FAST '19)*, pp. 173–190. USENIX Association, 2019. URL https://www.usenix.org/conference/fast19/presentation/bailleu.

Borja Balle and Yu-Xiang Wang. Improving the Gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *PMLR*, pp. 403–412, 2018. URL http://proceedings.mlr.press/v80/balle18a.html.

Borja Balle, Gilles Barthe, Marco Gaboardi, Justin Hsu, and Tetsuya Sato. Hypothesis testing interpretations and Renyi differential privacy. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 108 of *PMLR*, pp. 2496–2506, 2020. URL http://proceedings.mlr.press/v108/balle20a.html.

Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS '14)*. Internet Society, 2014. doi: 10.14722/ndss.2014.23212.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security '21)*, pp. 2633–2650. USENIX Association, 2021. URL https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting.

Tudor Cebere, David Erb, Damien Desfontaines, Aurélien Bellet, and Jack Fitzsimons. Privacy in theory, bugs in practice: Grey-box auditing of differential privacy libraries. *Proceedings on Privacy Enhancing Technologies*, 2026(3):467–483, 2026. URL https://petsymposium.org/popets/2026/popets-2026-0091.php.

Rishav Chourasia, Ergute Bao, Uzair Javaid, and Xiaokui Xiao. Auditing Apple's DifferentialPrivacy.framework: Implementation bugs, misconfigurations, and practical risks. In *2026 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2026. URL https://sp2026.ieee-security.org/accepted-papers.html.

Lynn Chua, Badih Ghazi, Pritish Kamath, Ravi Kumar, Pasin Manurangsi, Amer Sinha, and Chiyuan Zhang. How private are DP-SGD implementations? In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, pp. 8904–8918, 2024. URL https://proceedings.mlr.press/v235/chua24a.html.

Jinshuo Dong, Aaron Roth, and Weijie J. Su. Gaussian differential privacy. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(1):3–37, 2022. doi: 10.1111/rssb.12454.

Vasisht Duddu, Oskari Järvinen, Lachlan J. Gunn, and N. Asokan. Laminator: Verifiable ML property cards using hardware-assisted attestations. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy (CODASPY '25)*, pp. 317–328. ACM, 2025. doi: 10.1145/3714393.3726492.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014. doi: 10.1561/0400000042.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography (TCC 2006)*, pp. 265–284. Springer, 2006. doi: 10.1007/11681878_14.

Hubert Eichner, Daniel Ramage, Kallista Bonawitz, Dzmitry Huba, Tiziano Santoro, Brett McLarnon, Timon Van Overveldt, Nova Fallen, Peter Kairouz, Albert Cheu, Katharine Daly, Adrià Gascón, Marco Gruteser, and Brendan McMahan. Confidential federated computations. *arXiv preprint arXiv:2404.10764*, 2024. URL https://arxiv.org/abs/2404.10764.

Vitaly Feldman and Tijana Zrnic. Individual privacy accounting via a rényi filter. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 28080–28091, 2021.

Tomás González, Mateo Dulce-Rubio, Aaditya Ramdas, and Mónica Ribero. Sequentially auditing differential privacy. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, 2025. URL https://arxiv.org/abs/2509.07055. arXiv:2509.07055.

Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. Numerical composition of differential privacy. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pp. 11631–11642, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/6097d8f3714205740f30debe1166744e-Abstract.html.

Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation — a virtual machine directed approach to trusted computing. In *Proceedings of the 3rd Virtual Machine Research and Technology Symposium (VM '04)*, pp. 29–41. USENIX Association, 2004. URL https://www.usenix.org/conference/vm-04/semantic-remote-attestation%E2%80%94-virtual-machine-directed-approach-trusted-computing.

Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pp. 533–549. USENIX Association, 2016. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/hunt.

Intel Corporation. Intel trust domain extensions (Intel TDX) module base architecture specification. Intel document 348549, 2025. URL https://cdrdv2-public.intel.com/853286/intel-tdx-module-base-spec-348549006.pdf. Retrieved 2026-08-06.

Matthew Jagielski, Jonathan Ullman, and Alina Oprea. Auditing differentially private machine learning: How private is private SGD? In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/fc4ddc15f9f4b4b06ef7844d6bb53abf-Abstract.html.

Jiankai Jin, Eleanor McMurtry, Benjamin I. P. Rubinstein, and Olga Ohrimenko. Are we there yet? timing and floating-point attacks on differential privacy systems. In *2022 IEEE Symposium on Security and Privacy (S&P)*, pp. 473–488. IEEE, 2022. doi: 10.1109/SP46214.2022.9833672.

Jiankai Jin, Chitchanok Chuengsatiansup, Toby Murray, Benjamin I. P. Rubinstein, Yuval Yarom, and Olga Ohrimenko. Elephants do not forget: Differential privacy with state continuity for privacy budget. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*, pp. 1909–1923. ACM, 2024. doi: 10.1145/3658644.3670281.

Noah Johnson, Joseph P. Near, Joseph M. Hellerstein, and Dawn Song. Chorus: a programming framework for building scalable differential privacy mechanisms. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 535–551. IEEE, 2020. doi: 10.1109/EuroSP48549.2020.00041.

Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. Pesos: Policy enhanced secure object store. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*, pp. 25:1–25:17. ACM, 2018. doi: 10.1145/3190508.3190518.

Mathias Lécuyer, Riley Spahn, Kiran Vodrahalli, Roxana Geambasu, and Daniel Hsu. Privacy accounting and quality control in the Sage differentially private ML platform. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*, pp. 181–195. ACM, 2019. doi: 10.1145/3341301.3359639.

Tao Luo, Mingen Pan, Pierre Tholoniat, Asaf Cidon, Roxana Geambasu, and Mathias Lécuyer. Privacy budget scheduling. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI '21)*, pp. 55–74. USENIX Association, 2021. URL https://www.usenix.org/conference/osdi21/presentation/luo.

Xin Lyu. Composition theorems for interactive differential privacy. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback protection for trusted execution. In *26th USENIX Security Symposium (USENIX Security '17)*, pp. 1289–1306. USENIX Association, 2017. URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/matetic.

Microsoft. Secure key release with Azure key vault and Azure confidential computing. Microsoft Learn, Azure Confidential Computing documentation, 2023. URL https://learn.microsoft.com/en-us/azure/confidential-computing/concept-skr-attestation. Accessed 2026-08-18.

Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, pp. 650–661. ACM, 2012. doi: 10.1145/2382196.2382264.

Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pp. 263–275. IEEE, 2017. doi: 10.1109/CSF.2017.11.

Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Advances in Cryptology — CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pp. 126–142. Springer, 2009. doi: 10.1007/978-3-642-03356-8_8.

Ilya Mironov, Kunal Talwar, and Li Zhang. Rényi differential privacy of the sampled gaussian mechanism. *arXiv preprint arXiv:1908.10530*, 2019. URL https://arxiv.org/abs/1908.10530.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT* '19)*, pp. 220–229. ACM, 2019. doi: 10.1145/3287560.3287596.

Milad Nasr, Jamie Hayes, Thomas Steinke, Borja Balle, Florian Tramèr, Matthew Jagielski, Nicholas Carlini, and Andreas Terzis. Tight auditing of differentially private machine learning. In *32nd USENIX Security Symposium (USENIX Security '23)*, pp. 1631–1648. USENIX Association, 2023. URL https://www.usenix.org/conference/usenixsecurity23/presentation/nasr.

Joseph P. Near, David Darais, Naomi Lefkovitz, and Gary S. Howarth. Guidelines for evaluating differential privacy guarantees. Technical Report NIST Special Publication 800-226, National Institute of Standards and Technology, 2025.

Jianyu Niu, Wei Peng, Xiaokuan Zhang, and Yinqian Zhang. NARRATOR: Secure and practical state continuity for trusted execution in the cloud. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*, pp. 2385–2399. ACM, 2022. doi: 10.1145/3548606.3560620.

NVIDIA Corporation. Unified deployment guide for confidential computing. NVIDIA document DU-12302-001, 2026. URL https://docs.nvidia.com/cc-deployment-guide-tdx.pdf. Retrieved 2026-08-06.

Bryan Parno, Jacob R. Lorch, John R. Douceur, James W. Mickens, and Jonathan M. McCune. Memoir: Practical state continuity for protected modules. In *2011 IEEE Symposium on Security and Privacy (S&P)*, pp. 379–394. IEEE, 2011. doi: 10.1109/SP.2011.38.

Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H. Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Guha Thakurta. How to DP-fy ML: A practical guide to machine learning with differential privacy. *Journal of Artificial Intelligence Research*, 77:1113–1201, 2023. doi: 10.1613/jair.1.14649.

Ryan M. Rogers, Aaron Roth, Jonathan Ullman, and Salil Vadhan. Privacy odometers and filters: Pay-as-you-go composition. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.

Ahmad-Reza Sadeghi and Christian Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *Proceedings of the 2004 Workshop on New Security Paradigms (NSPW '04)*, pp. 67–77. ACM, 2004. doi: 10.1145/1065907.1066038.

Ali Shahin Shamsabadi, Gefei Tan, Tudor Cebere, Aurélien Bellet, Hamed Haddadi, Nicolas Papernot, Xiao Wang, and Adrian Weller. Confidential-DPproof: Confidential proof of differentially private training. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=PQY2v6VtGe.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (S&P)*, pp. 3–18. IEEE, 2017. doi: 10.1109/SP.2017.41.

Thomas Steinke, Milad Nasr, and Matthew Jagielski. Privacy auditing with one (1) training run. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/9a6f6e0d6781d1cb8689192408946d73-Abstract-Conference.html.

Raoul Strackx and Frank Piessens. Ariadne: A minimal approach to state continuity. In *25th USENIX Security Symposium (USENIX Security '16)*, pp. 875–892. USENIX Association, 2016. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/strackx.

Raoul Strackx, Bart Jacobs, and Frank Piessens. ICE: a passive, high-speed, state-continuity scheme. In *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC '14)*, pp. 106–115. ACM, 2014. doi: 10.1145/2664243.2664259.

Haochen Sun and Xi He. GPM: The Gaussian pancake mechanism for planting undetectable backdoors in differential privacy. *Proceedings of the ACM on Management of Data*, 4(3), 2026. doi: 10.1145/3802068. Also presented at ACM SIGMOD 2026; arXiv:2509.23834.

Shun Takagi and Satoshi Hasegawa. Securing private federated learning in a malicious setting: A scalable TEE-based approach with client auditing. *Proceedings on Privacy Enhancing Technologies*, 2026(1):486–508, 2026. URL https://petsymposium.org/popets/2026/popets-2026-0025.php. Also arXiv:2509.08709.

Pierre Tholoniat, Kelly Kostopoulou, Mosharaf Chowdhury, Asaf Cidon, Roxana Geambasu, Mathias Lécuyer, and Junfeng Yang. DPack: Efficiency-oriented privacy budget scheduling. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*, pp. 1194–1209. ACM, 2025. doi: 10.1145/3689031.3696096.

Tim van Erven and Peter Harremoës. Rényi divergence and kullback-leibler divergence. *IEEE Transactions on Information Theory*, 60(7):3797–3820, 2014. doi: 10.1109/TIT.2014.2320500.

Timon Van Overveldt and Daniel Ramage. Discovering new words with confidential federated analytics. Google Research Blog, 4 March 2025, 2025. https://research.google/blog/discovering-new-words-with-confidential-federated-analytics/.

Yu-Xiang Wang, Borja Balle, and Shiva Prasad Kasiviswanathan. Subsampled Rényi differential privacy and analytical moments accountant. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 89 of *PMLR*, pp. 1226–1235, 2019. URL http://proceedings.mlr.press/v89/wang19b.html.

Zheng Xu, Yanxiang Zhang, Galen Andrew, Christopher A. Choquette-Choo, Peter Kairouz, H. Brendan McMahan, Jesse Rosenstock, and Yuanbo Zhang. Federated learning of Gboard language models with differential privacy. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pp. 629–639. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-industry.60.

Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*, 2021. URL https://arxiv.org/abs/2109.12298. Privacy in Machine Learning (PriML) workshop, NeurIPS 2021.

Ruiyi Zhang, Lukas Gerlach, Daniel Weber, Lorenz Hetterich, Youheng Lü, Andreas Kogler, and Michael Schwarz. CacheWarp: Software-based fault injection using selective state reset. In *33rd USENIX Security Symposium (USENIX Security '24)*. USENIX Association, 2024.

Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, pp. 283–298. USENIX Association, 2017. URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zheng.

A Extended Background, Design, Evaluation, and Discussion

A.1 Background and threat model, extended

Where the stakes and the lineage come from. The main text states two points without support. The setting is not hypothetical: published ε values are routine in deployed systems (Abowd et al., 2022; Xu et al., 2023), confidential GPU modes extend the attested envelope across the PCIe boundary (NVIDIA Corporation, 2026), and NIST places ε at the apex of its privacy pyramid while locating "factors that can undermine a DP guarantee, such as lack of sufficient security" in the layer beneath (Near et al., 2025). And the observation that a measurement is not a statement about a property is two decades old (Sadeghi & Stüble, 2004; Haldar et al., 2004); what is new here is a setting in which the property is a regulatory artifact.

Figure 4 places P1–P5 relative to the attestation boundary, and says in a picture what the P1–P5 paragraph of Section 2 says in words: P1–P3 are per-step and stateless, so a per-iteration proof can bind them, while P4–P5 are properties of an execution, held in mutable state that outlives any measured process.

The accounting formalism, and what the composed state forgets. The main text states the Rényi accountant's transition and conversion; we add two details here. The per-step vector ρ is that of the sampled Gaussian mechanism (Mironov et al., 2019; Wang et al., 2019), and the conversion minimises the Rényi-to-approximate-DP bound over the order grid $\mathcal{A}$ at the target δ (Balle et al., 2020); other formalisms differ in the divergence they accumulate (Dong et al., 2022; Gopi et al., 2021) but not in this shape. And the composed state being a lossy digest of the touches that produced it is not a defect in the model the accounting literature works in, where the accountant is a pure function of the run — it becomes one only when the accountant is a service with a persistence layer somebody else administers.

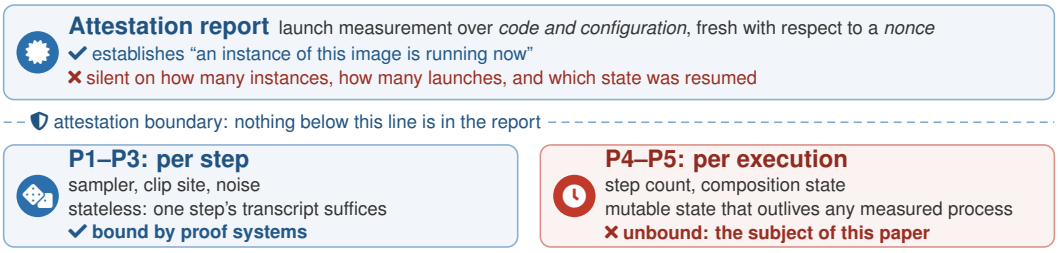


Figure 4: What the attestation report does not reach.

Deployment properties. Section 2 names snapshot, restore and fork as operations the infrastructure around every real accountant implements and no privacy analysis models. Three deployment properties turn them from features into exposure.

Restore is routine rather than exceptional, since long runs are scheduled on preemptible capacity: where restore is normal, rollback is normal, and an operator has no event to treat as suspicious. The accountant is one logical object over many processes, so whether one rank is authoritative or each accumulates locally determines whether restarting a replica loses a step, double-counts one, or forks the accumulator — a question of deployment topology that the accounting literature has no vocabulary for. And the accountant often outlives the job, becoming a long-lived, frequently multi-tenant service with its own persistence layer (Lécuyer et al., 2019; Luo et al., 2021; Johnson et al., 2020; Tholoniat et al., 2025) administered by the platform — precisely the party whose honesty attestation was introduced to avoid assuming.

The secondary adversaries. The *provider itself* has a compliance or competitive incentive to claim a smaller ε than it enforces. It controls the training infrastructure, the checkpoint storage and the scheduler, but must produce a report and a transcript that verify, because a claim that does not verify is worse for it than no claim. A third party matters though it attacks nothing: the *downstream exploiter* who mounts a membership-inference, reconstruction, or extraction attack against the released model (Shokri et al., 2017; Carlini et al., 2021), and who need not be the party that weakened the guarantee.

Surfaces, and generality. The learning algorithm is not where the attack surface is: the loader determines P1 and is a separate process in mainstream stacks, so the batch is an input to the measured code rather than a product of it; the randomness path determines P3; accumulation and packing determine P4 by fixing what a step means; and the checkpoint machinery holds the state P5 depends on, administered by the platform rather than the enclave. The result is architectural rather than an artifact of one implementation, because every mainstream DP stack separates the mechanism from the accountant — that separation is what makes an accountant reusable — and every confidential-training offering measures an image rather than a data stream. The attacks therefore transfer across TEE vendors, none of which attests a multiset of accounted steps, and across accounting formalisms, since rollback, forking and replay target the accumulator's state rather than the divergence it accumulates.

Out of scope. We do not attack the TEE: side channels, fault injection, firmware compromise, key extraction, defeats of memory encryption or integrity, and compromise of the attestation authority or the relying party's verification code are out of scope. Nor do we consider a provider that publishes an ε with no attestation at all — the assurance this paper shows to be manufactured is precisely the assurance such a provider never offered. We take the DP guarantee itself, and the question of whether commonly reported ε values are meaningful (Dwork & Roth, 2014; Ponomareva et al., 2023), as given; those debates are orthogonal to whether the reported value is the one the run produced.

A.2 Design details

The step record and the sealed beacon, in full. A *step record* is $r_i = (rid, j, i, c_i, q_i, \sigma_i, C_i, m_i, h_{i-1})$ with $h_i = H(r_i)$, naming the run, replica, index, a commitment to the batch, the mechanism parameters, and the hash of the preceding record on that replica; the shim signs a chain head every k steps. The batch for step i has its index set fixed by

$s_i = \mathrm{KDF}(\beta, \kappa(j, i))$, a function of its sample context: with independent replicas the replica identity j is in the derivation, so two replicas never silently draw one schedule; with shared contexts it is left out on purpose, and a second release on the shared sample is priced as a repeat (Algorithm 1). A retry reuses its own context rather than minting a new draw.

Who knows what, since the amplification rests on it. $\beta = H(\text{beacon}, \nu)$ takes two inputs with opposite jobs. The *beacon* is public randomness settled *after* the dataset commitment: it is what stops any party, the accountant included, from grinding a favourable schedule, and it is worthless as a secret because everyone sees it. The *nonce* ν is generated inside the accountant's confidential VM, never leaves it, and is what keeps β unpredictable to the party the guarantee is claimed against; the accountant publishes $H(\nu)$ at registration, which pins ν before the beacon is settled so it cannot be chosen in response to one, and publishes $H(\beta)$ afterwards. Neither alone suffices: a public beacon with a public nonce is a public schedule, and a secret nonce with no beacon is a schedule the accountant could grind. Publishing β, so that any verifier could re-derive the intended batch without holding the data, is the tempting alternative; it makes the sample public, forfeits the amplification, and inflates the accounted ε by nearly four orders of magnitude over a run (Table 30). What the verifier gets in place of the sample is the accountant's measurement. A deployment unwilling to make that trade sets `public=True` and is charged the *unamplified* mechanism, the correct price of a public sample.

Two further details of the commit protocol. Beyond the ordering and the conditional advance, the advance is also *checked* against suppression (line 3 of Algorithm 1), because a host that can rewind a counter can more cheaply freeze one, and every value a frozen counter returns is one it really held. And recovery never reinstates a stale Σ as authoritative: it restores the register's accounting frontier over it (Algorithm 2), or refuses (Section A.2). A sealed counter, or a sealed n or s, *ahead* of the register means the register itself was rewound, and there is nothing to do but refuse.

Why recovery reconstructs the accounting instead of bounding it. A seal is a statement about a moment, so a quantity that an older seal can replace has to be recovered from somewhere the adversary cannot reach. Bounding what lost generations could have done does not work: recovery raises $D.\mathrm{high}$, the next reservation sets $b_r \leftarrow D.\mathrm{high}[r] + k'$ *from the raised value*, and two surcharge-only commits hold b_r still while the generation counter runs — so no bound linear in the generation gap is closed under RESTART → AUTHORISE → RESTART. The register's frontier, below, reconstructs instead; two weaker rules, and the schedules that defeat them, are rows 2–3 of Section A.3.

Who owns a prepaid unit. The two quantities a reservation creates have *different owners*, and a reader who merges them gets a protocol that does not hold. Prepaid capacity w is a **run-level pool**: any registered replica may spend a unit from it, and w is one scalar because the charge is one charge. The possible-index bound is **per replica**: a reservation of k' units seals $b_r \leftarrow D.\mathrm{high}[r] + k'$ for *every* registered r (line 17), not only for the replica whose request triggered it, because any of them may spend that pool before the next commit. Both recovery paths then burn every replica's mark — the exact path to b_r (line 10), the reconstructing path to $\max(b_r, B^\star)$ (line 12).

Sealing only the requesting replica's bound would be unsound, and the failure is short: with two replicas and $k = 2$, A's first release reserves two units and seals only b_A; B spends the remaining pooled unit with no commit of its own; a restart restores $D.\mathrm{high}[B]$ from a blob that never recorded B's range, and B's next release at an index it has already used is classified *fresh* and priced $\rho(q, \sigma)$. The count stays conservative while the price does not, which is exactly the failure Proposition 1 rules out. Burning every replica is an over-approximation when the window was not in fact shared — it forfeits indices nobody used — and that is the safe direction. Section A.4 runs the sequence against the implementation.

The accounting frontier. The register already carries a payload, the lineage digest. It also carries the *accounting frontier* $F(\Sigma) = (\max_j b_j,\ n,\ s)$: the largest index any sealed reservation could have released, the durable accounted count, and the number of unamplified surcharge units charged. All three are nondecreasing over a run, and the register refuses an advance that would lower any of them — the same assumption already made about the counter, extended from one small integer to three. Recovery then *reconstructs*: it burns to $B^\star$ and restores $(n^\star, s^\star)$, which restores R exactly, because

$$R \;=\; n\,\rho(q, \sigma) \;+\; s\,\Delta \qquad \text{is an invariant of the protocol.}$$

Every reservation adds units to n; every surcharge adds Δ and increments s; nothing else touches R. The invariant holds because the mechanism parameters are pinned at registration, which AUTHORISE *enforces* — a record offering a different σ is refused rather than priced, since pricing it would make (n, s) an incomplete description of R and put recovery back to bounding.

The schedules that defeat weaker rules. Two schedules defeat rows 2–3 of Section A.3, and both pass under the frontier. Rolling back the generation that carried a surcharge requires 2.0000000 at order two. Recovering from the latest blob alone yields 1.4×10^{-5}, a per-generation envelope over-recovers, and the frontier recovers 2.0000034: the accounted cost exactly, including the window's prepaid units. Restoring a blob sealed before a later reservation burns through index 11, so the index that reservation released comes back classified duplicate. The latest-blob positive control costs 0 extra, which is the property a conservative repair is most likely to break.

The composed schedule, run against the implementation. Its $k = 2$ instance assumes a recovery that does not commit. The implementation *does* commit on recovery, which inflates the later generation gap, so that instance happens to pass even under a per-generation envelope; `state_machine.json` records it releasing 4 distinct outputs, 0 of them above the recovered bound. That does not vindicate the envelope, because a $k = 3$ instance defeats it. Two surcharge-only commits hold the sealed bound flat while the generation counter advances; restoring the older blob then moves the bound by $2k$, the next window releases three indices above it, and a later restore of an intermediate blob burns short of them. Under the envelope the checker reports exactly that — released indices above the recovered bound, which would be priced as fresh subsamples. Under the frontier it reports 6 distinct outputs, a recovered order-two vector of 3.000005 against a required 3.000005, a recovered bound of 5 and 0 indices above it. The 19-row event log — generation, register frontier, high-water mark, each re-offered index's actual disposition, and the durable cost after every event — is in `state_machine.json`, and the invariant is checked both after the last release and after a further latest-state recovery, because passing at authorisation alone is not the property.

The register interface, stated as an interface. The construction asks for more than a bare monotonic counter, so we state the register's requirements in one place. The register is a service with exactly two operations, and the assumptions are on *them*, not on a hardware datasheet.

READ$() \to (a, d, B, n, s)$ returns *one* snapshot. It must be atomic: a reader must not observe a from one transition and (B, n, s) from another, because recovery compares a blob's generation against a and then reconstructs from (n, s), and a torn read would let it restore a frontier that never coexisted with that generation. The snapshot is authenticated under a key the register and the accountant share. On the quorum service it is the tuple a majority has reached; on a TPM the adapter of Section C.5 would have to read its counter and data indices as one snapshot, which is part of what that unbuilt adapter must supply.

Authentication is not freshness. An *authentic* reply captured from an earlier read verifies perfectly when replayed, so a host that records one and serves it later defeats an authentication-only check with no forgery. The contract therefore requires every reply to be **fresh and bound to the request**: the reader sends a nonce, the register MACs that nonce, a digest of the exact request, the operation and (a, d, F), and the reader rejects any reply that does not name its own request. A CAS success is thereby bound to the generation it advanced from and to the digest and frontier it was asked to write, so a replayed success cannot be read as a success for a different transition.

CAS(from $a,\ d',\ F') \to a'$ advances to $a + 1$ only from a, and only if F' dominates the held frontier coordinatewise; it writes $(a{+}1, d', F')$ as one transition. That atomicity is the linearisation point of the whole protocol: every durable state change in Algorithm 1 happens exactly when its CAS returns, and two accountants racing at one generation are ordered by which CAS the register took first.

Failure is a third outcome, not an exception. A CAS that is refused — because another holder advanced, because the frontier would regress, or because a quorum is unreachable — latches $closed$ and mints nothing (Algorithm 1, line 3). A READ that cannot be completed — unreachable, or answered by a reply that fails the freshness check — leaves RESTART with no frontier and it refuses (line 11). *Neither failure is distinguishable from an adversary's refusal, and neither needs to be*: both stop the run, which is the direction in which stopping is free.

What the platform must supply, and what it need not. It must supply monotonicity in a and in each coordinate of F, atomicity of the two operations, authentication *and freshness* of the response —

bound to the request, so a replayed authentic reply is rejected, with the reader's nonce drawn from a source no rollback restores — and per-tenant isolation. It need not supply confidentiality — F counts transitions and units, not records (Section B.1 lists it in the adversary's view) — and it need not store more than a generation, a 32-byte digest and three integers. Section C.5 maps this onto the facilities that exist, and Table 24 attacks a register that meets part of it.

What is built. `QuorumAnchor` binds every reply to its request — each replica MACs the exact request line, which carries the operation, the tenant, the expected generation, the digest, the frontier and a fresh nonce — and every socket timing in this paper was re-taken on the authenticated protocol. The same contract is implemented in process as `RegisterService` and `FreshRegisterClient`, and both are exercised against a host that forwards, records, withholds and replays *authentic* replies (`ReplayingProxy`; Table 24, A11–A16). Each position also runs a weaker client that the host must fool, or the refusal would prove nothing:

- **A11**, an old READ reply served after a store rollback: a MAC-only client accepts the stale blob as current and recovers 10 accounted units after 40 were minted; the fresh client refuses.
- **A12**, an old CAS success served to a clone whose own request asks for another frontier ($s{+}2$, from a re-executed index): the MAC-only clone mints while the register holds a frontier without its surcharge; the fresh clone latches and mints nothing.
- **A13**, a reply withheld when produced (the register looks unreachable) and delivered after later requests: accepted by the MAC-only client, refused by the fresh one.
- **A14**, a nonce derived from the accountant's own sealed state — the natural way to make nonces "never repeat" without a random source. After a rollback of that state the restart re-issues the very request whose reply the host kept, and even full request binding accepts it. The nonce must therefore come from outside every state the adversary can restore; ours is drawn from the platform generator, and A14's implementation row refuses.
- **A15**, A11 on the *socket* register the timings use: the host sits below authentication and answers a new READ with every replica's old reply. The unauthenticated protocol accepts them; the authenticated one counts each as no reply, loses its quorum and refuses.
- **A16**, the *plane's* own read: after a rollback of the record plane's memory, its restart is answered with an old authentic reply, which would hand it an old generation floor. A MAC-only plane then serves a token from an early window a second time — a batch for a step nobody charged; the fresh plane refuses the reply and serves nothing.

The binding costs 24 µs per register call in process (serialisation, nonce and two HMACs; host load 23); the socket timings of Figure 3(b) include it, and at one register call per window it is small beside the round trip it sits inside. Freshness across a byte-for-byte clone of *live* memory — generator state included — is condition (vii)'s, not this contract's.

Three consequences, one of them a cost. First, the proof needs no induction over generations: the frontier is monotone, outside the rollback, and equal to its value at the last commit, so composition has nothing to accumulate. Second, recovery does not over-charge: a per-generation envelope would make a rollback cost a whole unamplified step per generation, whereas the exact reconstruction does not, so a run that survives a rollback keeps its budget. Third, the construction asks something of the platform: a register that *cannot* carry a payload leaves recovery from a stale blob with nothing sound to reconstruct from, and RESTART therefore *refuses* (Algorithm 2, line 11). That is a real availability cost on such a platform and it is the correct direction; Section C.5 says which facilities can carry it, and Table 24 attacks a register that cannot.

What an ordinary crash costs, precisely. A restart that finds the state the register names — because the host merely crashed and still holds it — adds *no charge at all* (Algorithm 2, line 10). That is not the same as costing nothing: the outstanding reservation is forfeited on *every* restart, clean or not (line 9), which is the utility price Table 8 measures. "Free recovery" below means no additional accounting charge, never no forfeiture. Conversely, a commit torn between its advance and its seal loses the exact latest state with no adversary present, so a missing authenticated state is an *operational condition* and not evidence of intent; what distinguishes the cases is whether the register's frontier is available, not why the blob is missing.

Algorithm 1 gives COMMIT and AUTHORISE in both replica modes, and Algorithm 2 gives RESTART, which reconstructs the accounting from the register's frontier rather than bounding it and refuses

when the register cannot carry one. The repeat test and the upgrade key on the sample context κ, not on the replica: an earlier printing keyed both per replica, and Section A.4 runs that rule as a failing control.

Why a bare increment cannot replace compare-and-swap. Section 3.1 makes the anchor advance conditional. A bare increment cannot do the same work. Two accountants cloned from one sealed blob — the case a launch measurement definitionally cannot exclude, reporting that an instance runs and never how many — each ask for a fresh value, each receive one larger than they held, each conclude they advanced, and both seal divergent histories. Compare-and-swap from the generation the caller believes it holds lets exactly one win, and binding a digest of the sealed state to the transition makes the register a lineage record rather than a tally.

This statement concerns clones that start from a sealed blob. A live snapshot made after a reservation would copy the in-memory w and is outside the reachable-state model unless the platform supplies a single-live-instance lease; deployments unable to exclude that operation use $k = 1$. Launch measurement alone does not supply this property, and we do not treat it as if it did (Section 2).

The error state after a failed commit, across calls and not only within one. Refusing to return a capability when the durable write fails (Algorithm 1) settles one call and leaves the next one unspecified, which is not enough: the routine has by then already added a window to the in-memory R and n. The contract is three statements, and it is part of the state machine rather than only of this prose: the $closed$ flag of Algorithm 1, set at line 4, tested at line 7, and cleared only by Algorithm 2. An implementation that gets any of it wrong is detectable only by a test that makes a second call. The failure *latches*: every subsequent AUTHORISE is refused, without looking at the record, until a recovery transition succeeds. Recovery is the only thing that clears the latch, and it clears it by *reloading durable state*, so the mutation that was never sealed is discarded rather than inherited — which also means the failed call's window is not silently spendable afterwards. And the refusal costs nothing, because no capability was minted: the step did not run. The regression accordingly injects a commit failure, asserts the call mints nothing, makes three more authorisation attempts *before* any recovery and requires all three to be refused, and then checks that the post-recovery vector equals the last durable one rather than the one the failed call left in memory.

Multi-tenancy. Anchors and sealed state must be per-tenant. A single shared anchor advanced by a busy tenant would mask another tenant's rollback: the victim's restart would find a register comfortably ahead of its sealed state and read that as its own progress. This is easy to miss because a shared counter is the cheaper deployment, and it is wrong for a reason that has nothing to do with contention.

Duplicates are charged, never refused. D records the highest authorised index per replica. Fresh indices are contiguous (condition (vi)), so every index at or below a replica's mark has been authorised on it, and a record whose sample context lies at or below the marks of the replicas sharing that context (line 12) is charged and logged as a duplicate, never rejected, because the re-executed step really did touch the data (Table 19). Contiguity also makes an exception list of indices committed out of order unnecessary: such a list would stay empty, so the state keeps none. Refusing at the *gate* is legitimate, since a gated step has not yet run — the asymmetry is deliberate, and an implementation that refuses duplicates at the accountant would be the natural one to write and a privacy bug.

The commit window, stated as a cost. Writing the charge ahead moves the commit window's cost from privacy to utility rather than removing it. Because a window of k steps is charged before any of it is spent, a crash forfeits the unspent remainder, up to k steps per crash, so a trainer that crashes often exhausts its declared budget early and is refused at the gate.

Retention, and why a coverage cap does not work. Against adversary A1 of Section 2 retention is a buffer-sized window; A2 retains everything, and for A2 the exposed fraction is not $1 - (1 - q)^W$ for a prefetch depth W but $1 - (1 - q)^t$ in the steps served so far. The knob that would bound it — refuse a token once the union of served indices exceeds a declared fraction of the corpus — does not survive contact with DP-SGD, because coverage saturates: even a cap of 0.999 allows 25% of the honest run. What does work is the gradient-releasing plane of Section 3.2, where W stops being a

parameter rather than being made small. Its marginal cost is 25 µs, negative whenever the trainable parameters are smaller than the batch (Section C.2).

Join, and how parallelism is declared. Parallelism is declared at run registration as a set of replica identities — Section C.3 works the declaration through gradient accumulation, packing and data parallelism — and the budget certificate is signed only over a state holding a head for every declared replica and having consumed each replica's chain up to that head. A run that silently spawns an undeclared replica has no registered identity for it, and its records are refused rather than accounted against someone else's branch. A replica declared *mid-run* is registered by a commit that seals its bound as $b_r \leftarrow w - 1$: it may spend from the open pool before the next commit, and its fresh indices start at zero, so that is what it can have released. Sealing $b_r = -1$ instead would let an exact recovery restore the joiner's mark below indices it has already used (`test_a_replica_joining_mid_window_cannot_reopen_its_indices`).

What the certificate asserts: one table, four layers. A reader should not have to assemble the claim from five appendices, and the paper's own claim is easy to overstate by conflating its layers. Table 2 separates them. Each row names what is claimed, what it assumes, and what kind of evidence stands behind it; no row inherits a conclusion from the row above except as the table says. Only the first row is demonstrated on the artifact; the second is proved about an idealised experiment; the third transfers the second to the deployed transcript under named computational assumptions; and the fourth says what actually ran.

Two scopes deserve their own sentences because they are the ones most easily lost. **The schedule**: the privacy rows assume one fixed before the run and non-interleaved in the order outputs are released — the number of releases, their contexts and their parameters — or one whose queries are fixed in advance. Under priced repeats that is a *scope*, which the verifier cannot check (Section B.1); a fresh-once registration makes it an enforced property the certificate names (Corollary 2). The training results meet it trivially — no context repeats — while the replay row of Table 4 and the checker's re-execution campaigns do not and are bookkeeping evidence (Section D.2). Certifying the *realised* cost of a run whose length depends on its own outputs would need a Rényi filter, whose hypothesis — a per-round conditional bound — fails here, as the counterexample of Section B.1 shows; fresh-once avoids the need by certifying the committed length instead. **The register**: without one that carries the frontier, recovery from a rolled-back store refuses (Section A.2), and without fresh, request-bound replies a replayed authentic reply can make a stale blob look current (Section A.2, A11–A16).

The certificate. On completion the accountant signs a certificate binding the dataset commitment, the run identity, the final chain head of every replica, the per-mechanism accounted step counts, the anchor value, (ε, δ), and its own measurement. A verifier checks the accountant's attestation report and that the certificate names the accountant it expects, checks anchor freshness, verifies the chain heads, and recomputes ε from the per-mechanism counts. It never replays training: Rényi composition is additive and each step's contribution depends only on (q, σ), so recomputing ε costs one Rényi vector per distinct mechanism rather than one per step. A MAC rather than a signature is sufficient for the release token and an order of magnitude cheaper (Table 29), because the token travels one hop between two parties who share a key and is never verified by a third.

What the certificate does not do is prove the model's provenance. It certifies a transcript; it does not by itself establish that a supplied set of weights was derived only from that transcript, and adding a digest of the weights would not establish it either — a digest binds *which* artifact is claimed, not *how* it was produced. What carries provenance is the permitted information flow, and only in the primary configuration of Section 3.2: if the plane is the sole holder of plaintext records and of the mechanism, every quantity that ever depended on the data reached the trainer as one of the plane's releases, each covered by a durable reservation (Proposition 1), so any model the trainer produces is a function of those releases and of data-independent state. A relying party that cannot enforce sole custody — the record plane, where raw records cross — gets the weaker statement: the certificate covers fresh *access*, and what a trainer did afterwards with records it was given is outside it. Which of the two a certificate carries is its *mode* field, and that is the field a verifier reads first.

The residual, and three limits of the design. Retention is the inherent limit of gating data at all. Against A1 it is a buffer-sized window; A2 retains everything, and the exposed fraction is $1 - (1 - q)^t$

in the steps served so far, which DP-SGD saturates by construction (Table 22: 95% of the corpus after 3,000 steps). What works is the gradient-releasing plane, where only the noised gradient crosses the boundary and the trainer never holds a plaintext record; the work is relocated rather than added, at a marginal 25 µs, and in Table 6 it trains the same 65 steps to 77.9% against the record plane's 77.8%. Three further limits belong to the design rather than to a limitations section. The anchor is the root of trust for the whole construction, and where no rollback-resistant counter exists DP-ATTEST degrades from prevention to detection (Section C.4). Nothing here binds hyperparameter honesty or dataset construction: a dataset in which one user contributes unboundedly many examples yields a certificate that is true and a guarantee that is not what a reader will take it to mean. And compromise of the accountant's own measurement defeats the design entirely, which is why the accountant is small.

As built. Sealed storage is AES-GCM under a key we hold — like real sealing, no freshness. One further substitution touches the mechanism rather than the harness: the gradient plane derives its noise seed as $H(\text{sealed seed, capability})$ and then draws from NumPy's PCG64. Determinism is what Proposition 1 needs and PCG64 gives it, but the DP guarantee needs a draw computationally indistinguishable from Gaussian to a party holding the outputs, which PCG64 does not give; a deployment keys a cryptographic DRBG from the same derived seed. That changes every byte of the noise and no property proved here. Both DP-SGD paths — the plane's on the gradient plane and the trainer's on the record plane — divide the noised sum by the *public* expected batch size qN, and the gradient plane releases the gradient and nothing else. Dividing by the *realised* size, or releasing it, would not implement the sampled Gaussian mechanism that (I2) describes: under Poisson sampling the realised size is a function of the inclusion bits, the kind of public value that the scalar example of Section B.1 shows can spend amplification. Every training result in this paper uses the public normalisation (`test_the_gradient_plane_divides_by_the_public_expected_batch`). The accountant is ours rather than a library's, because the attacks drive a state machine that library accountants deliberately do not expose, which is itself part of the argument. Correctness is checked by 110 tests, against closed forms and against the pitfalls of Section A.3.

Algorithm 1: Write-ahead commit and release (Section 3.1).

State : $\Sigma = (R, n, s, h, D, U, a, w, b)$, sealed; window k; anchor A

State : w: prepaid units left, a *run-level pool* any registered replica may spend; b_r: per-replica upper index bound of the sealed window; s: surcharge *units*, so $R = n\rho(q,\sigma) + s\Delta$ after every line

State : $\kappa(j,i)$: the *sample context* of index i on replica j — (rid, i) if replicas share contexts, (rid, j, i) if independent; $G(j)$: the replicas sharing j's contexts (all, or $\{j\}$); U: contexts whose first release has been upgraded; $\mathrm{com}_\beta(\kappa)$: the batch commitment of the sample the sealed seed β derives for κ (Section B.1)

State : $\Delta := \big(\rho(1,\sigma) - \rho(q,\sigma)\big)^+$; $\widehat{\Sigma} := H(\Sigma \setminus a)$, the lineage digest; $F(\Sigma) := (\max_j b_j,\ n,\ s)$, the *accounting frontier*

State : *closed*: latched by *any* failed durable transition, cleared only by RESTART

procedure COMMIT(Σ)**:**
- $a' \leftarrow A$.CAS(from $\Sigma.a$, $\widehat{\Sigma}$, $F(\Sigma)$) `// fails if another holder advanced, or if F would regress`
- **if** CAS *failed* ***or*** $a' \le \Sigma.a$ **then** *closed* $\leftarrow \top$; **return** REFUSE `// refused or suppressed: both latch`
- $\Sigma.a \leftarrow a'$; **if** seal(Σ) *fails* **then** *closed* $\leftarrow \top$; **return** REFUSE
- **return** OK

procedure AUTHORISE(r_i)**:**
- **if** *closed* **then return** REFUSE `// latched until RESTART reloads durable state`
- **if** $(q_i, \sigma_i) \notin \{(q,\sigma), (1,\sigma)\}$ **then return** REFUSE `// parameters pinned: what makes (n, s) describe R`
- **if** r_i.prev $\neq h[r_i.j]$ ***or*** $c_i \neq \mathrm{com}_\beta(\kappa(j,i))$ **then return** REFUSE
- **if** $i > D.\mathrm{high}[j] + 1$ **then return** REFUSE `// fresh indices are contiguous per replica`
- **if** $w = 0$ ***and*** $n \ge n_{\max}$ **then return** EXHAUSTED `// only when a new reservation is needed`
- `// Strict: T releases on one context total` $T\rho(1,\sigma)$. `Both tests key on the context, over authorisations: held capabilities count.`
- $d_i \leftarrow \mathbf{1}\big[i \le \max_{r \in G(j)} D.\mathrm{high}[r]\big]$; $up_i \leftarrow d_i \wedge (\kappa(j,i) \notin U)$; $m_i \leftarrow d_i + up_i \in \{0,1,2\}$ `// surcharge units this release owes`
- **if** $d_i = 1$ ***and*** *the run is registered fresh-once* **then return** REFUSE `// a used context: refused, not priced`
- **if** $w = 0$ **then**
  - $k' \leftarrow \max\{t \le \min(k, n_{\max} - n) : \varepsilon(R + t\rho(q,\sigma) + m_i\Delta, \delta) \le \varepsilon_{\max}\}$ `// complete prospective cost, upgrade included`
  - **if** $k' = 0$ **then return** EXHAUSTED
  - $R \mathrel{+}= k'\rho(q,\sigma) + m_i\Delta$; $n \mathrel{+}= k'$; $s \mathrel{+}= m_i$; $w \leftarrow k'$; $b_r \leftarrow \max\big(b_r,\ D.\mathrm{high}[r] + k'\big)$ **for every registered** r `// never lower a sealed bound: k' shrinks at the budget's edge`
  - **if** $m_i > 0$ **then** $U \leftarrow U \cup \{\kappa(j,i)\}$
  - **if** COMMIT(Σ) $\neq$ OK **then return** REFUSE `// no capability after a failed durable write`
- **end**
- **else if** $\varepsilon(R + m_i\Delta, \delta) > \varepsilon_{\max}$ **then**
  - **return** EXHAUSTED
- **end**
- **else if** $m_i > 0$ **then**
  - $R \mathrel{+}= m_i\Delta$; $s \mathrel{+}= m_i$; $U \leftarrow U \cup \{\kappa(j,i)\}$; **if** COMMIT($\Sigma$) $\neq$ OK **then return** REFUSE `// seal the surcharge before mint`
- **end**
- $w \leftarrow w - 1$; $h[r_i.j] \leftarrow H(r_i)$; $D.\mathrm{high}[j] \leftarrow \max(D.\mathrm{high}[j], i)$
- **return** $\mathrm{MAC}_{key}(rid, j, i, c_i, u_i, \Sigma.a)$ `// the release token;` u_i: `the query charged`

Algorithm 2: Recovery from the register's accounting frontier.

State : $\Delta := \big(\rho(1,\sigma) - \rho(q,\sigma)\big)^{+}$, one surcharge unit

procedure RESTART():

- $(a^\star, d^\star, B^\star, n^\star, s^\star) \leftarrow A.\text{read}()$ `// ONE authenticated snapshot; see Section A.2`
- $\Sigma \leftarrow \text{unseal}()$; $\widehat{\Sigma}_0 \leftarrow H(\Sigma \setminus a)$ `// taken` *before* `any mutation: the digest covers` w
- **if** $\Sigma.a > a^\star$ ***or*** $\Sigma.n > n^\star$ ***or*** $\Sigma.s > s^\star$ **then return** REFUSE `// register rewound`
- **if** $\widehat{\Sigma}_0 \neq d^\star$ ***or*** $\Sigma.a \neq a^\star$ **then**
  - $\Sigma' \leftarrow \text{fetch}(d^\star)$ **if the platform serves it**
  - **if** Σ' ***and*** $H(\Sigma' \setminus a) = d^\star$ ***and*** $\Sigma'.a = a^\star$ **then** $\Sigma, \widehat{\Sigma}_0 \leftarrow \Sigma', d^\star$ `// verified, not trusted`
- **end**
- $w \leftarrow 0$ `// forfeit the outstanding reservation, always`
- **if** $\widehat{\Sigma}_0 = d^\star$ ***and*** $\Sigma.a = a^\star$ **then** $D.\text{high}[r] \leftarrow \max(D.\text{high}[r], b_r)$ for every r; *closed* $\leftarrow \bot$; **return** `// exact: this` *is* `the named state`
- **if** $(B^\star, n^\star, s^\star)$ *absent* **then return** REFUSE `// nothing sound to reconstruct from`
- $D.\text{high}[r] \leftarrow \max(D.\text{high}[r],\, b_r,\, B^\star)$; $b_r \leftarrow D.\text{high}[r]$ for every r `// re-derive` b `from the burnt mark, or the next` COMMIT `proposes a frontier below` $B^\star$ `and is refused`
- $R \mathrel{+}= (n^\star - n)\rho(q,\sigma) + (s^\star - s)\Delta$; $n, s \leftarrow n^\star, s^\star$; $\Sigma.a \leftarrow a^\star$; *closed* $\leftarrow \bot$ `// the only thing that clears the latch`
- COMMIT(Σ) `//` **`reconstruction commits; the exact path at line 10 does not`**

`//` U `is whatever the loaded blob carried, possibly` *less* `than was paid. A later first repeat on such a context then pays the upgrade again, which over-charges and is the permitted direction;` s `rises with it, so the identity above still holds. Every` $D.\text{high}[r]$ `is now at least` $B^\star$`, so every context released before the crash tests as a repeat in either mode.`

Table 2: The guarantee, layer by layer.

Layer	Statement	Assumes	Evidence
Bookkeeping (Proposition 1)	At every reachable state of *every* interleaving of authorisations, releases, crashes, restores and repeated redemptions: $A \geq M$, and $R \succeq \sum_o \rho(o)$ with ρ the strict price per sample context	(i)–(vii): a monotone, fresh register carrying the frontier; sole custody; query-bound capabilities; capability-keyed noise; write-ahead surcharges; contiguous clocks; no live-RAM clone in a window	Proof (Section B.1); enumeration and named traces against the implementation (Section 4.3 and Table 9)
Ideal privacy (Corollaries 1 and 2)	The Rényi divergence between the ideal views of neighbouring D' and D is at most $\sum_\kappa c_\kappa$, the strict charge, at every order; under fresh-once, at most $T^\star \rho(q, \sigma)$	(F) mechanism fidelity; (P) hiding commitment; a schedule *fixed* before the run and *non-interleaved in release order* (or with queries fixed in advance). **Enforced** by a fresh-once registration, which the certificate names; a deployment obligation, unchecked, under priced repeats	Proof. For fresh-once, also the enumeration's fresh-once campaign and its priced control (Section 4.3); a run staying under a threshold is not evidence for the bound (Section B.1)
Computational transfer	$(\varepsilon, \delta + \mathrm{negl})$ computational DP for $\mathrm{view}^{\mathrm{real}}$	(S*) over five disjoint oracle families (Section B.1), in the ROM; HMAC is a PRF; (iv) the noise generator is a PRG	Hybrid argument (Section B.1); not reduced for the artifact, whose PCG64 noise does not instantiate (iv)
Prototype	What ran: CPU processes, a software register (in process and over an authenticated socket), AES-GCM sealing, PCG64 noise	—	Every timing and trace in this paper; no confidential VM, attestation report, hardware sealing or TPM (Section C.2)

A.3 Design pitfalls and their regression tests

The specification above is stated once, in its current form. Table 3 lists natural variants of it that fail, with the schedule or check that exposes each failure and the rule that avoids it. Every row is pinned by a test, which either runs the variant as a failing control or checks that the specification avoids it.

The pattern. Rows 2, 3, 5, 7 and 8 reason about the *latest* sealed state, or about what a bounded number of lost transitions could have done, whereas the adversary's move is to make that boundary itself movable; the register's frontier is what the adversary cannot move. Rows 10, 12 and 13 conflate two of the three identities — replica, sample context, capability — that the specification keeps apart. Rows 1, 11 and 15 trust a label where the bytes say otherwise, and row 17 trusts a premise that the certificate does not establish. The register variants share one shape: each is a freshness test that compares the wrong pair of values. Checking that a counter is not behind misses a counter that was stalled; checking that it advanced misses who advanced it; checking a local anti-replay set misses that the set is itself rewindable.

Table 3: Natural variants of the protocol that fail, and the rule and test that replace each.

	Variant	How it fails	Rule used, and its test
1	One capability redeemable at any query; noise from an advancing stream	one charged window of k buys 41 distinct outputs	query inside the MAC; noise a PRF of the capability; *replay* verb
2	Surcharge added in memory and sealed with the next window; or sealed, but recovered from the latest blob only	an equal-generation reload, then a rollback of the sealing blob, drops it	surcharge sealed before mint (condition v); register frontier
3	Per-generation envelope $k\rho + \Delta$ and burn $b_j + gk$	not closed under RESTART $\rightarrow$ AUTHORISE $\rightarrow$ RESTART	recovery *reconstructs* from $F = (B, n, s)$; `composed_k3`
4	A commit without the s update, the strict upgrade, or the latch on a refused CAS; lineage digest taken after zeroing w	$R = n\rho + s\Delta$ fails; the recovery check breaks	Algorithm 1 as implemented; transition-map rows
5	b_j sealed for the requesting replica only	a pooled unit spent by another replica comes back fresh	b_r for every replica; `mixed_replica_traces` control
6	A per-round conditional bound for adaptive releases; `amplified-first` pricing at every order	false: $30.71 > 25$ at $q = 0.1$, $\sigma = 0.2$	not used; the group bound instead (Section B.1)
7	Recovery leaving b below the burnt mark	the next commit proposes a regressing frontier	$b_r \leftarrow D.\mathrm{high}[r]$ after the burn; continuation trace
8	A shortened final window lowering a sealed bound	safe, but the run latches closed with budget left	$b_r \leftarrow \max(b_r, \cdot)$; availability traces
9	Index j's batch treated as independent of every release at every other index	false for output-dependent queries	condition on the past; non-interleaving hypothesis
10	Shared-context replicas charged once, as one noised object	the plane releases two independently noised full gradients	repeat keyed by context; byte-level check
11	An authenticated register reply taken as fresh; a TPM NV index or a guest vTPM counter taken as the register	a replayed authentic reply verifies; neither is a tuple CAS	request-bound replies on both register paths (A11–A16); vTPM a counter-only register; TPM adapter named, unbuilt
12	Repeat test and upgrade set kept per replica	undercharges a shared context (3.4×10^{-6} against a strict 2)	one rule keyed by sample context; Table 9 control
13	A checker oracle keyed by (replica, index) or by (index, query)	blind to two replicas reading one sample; misprices independent replicas	an oracle keyed by sample context over the plane's release log (Table 10)
14	A mid-run joiner's bound sealed at -1	an exact recovery can reopen indices it has used	bound $w - 1$; `test_a_replica_joining_mid_window_cannot_reopen_its_indices`
15	DP-SGD normalised by the *realised* batch size, which the gradient plane also releases	a function of the inclusion bits, so not the sampled Gaussian mechanism	divide by the public qN; release nothing else
16	The record plane's generation floor applied on the gradient plane	a capability held across another replica's commit is refused after it has been paid for	no floor on the gradient plane; A9's floor on the record plane
17	A certificate that asserts the corollary's schedule premises without enforcing them	certifies bookkeeping, yet reads as a privacy statement	fresh-once registration: repeats refused, schedule committed, premises enforced and named in the certificate; fresh-once campaign and priced control; `test_fresh_once_*`
18	No commitment to the nonce before the beacon is settled	the accountant can choose ν after seeing the beacon, and so grind the schedule	tagged nonce commitment, published at registration and certified; `test_the_nonce_is_committed_under_its_own_tag`

A.4 EVALUATION DETAILS

Table 4 re-runs every attack against DP-ATTEST as implemented. Both *recomputed* columns are given because DP-ATTEST changes what can run, so a row's halves describe different executions; the undefended accountant reports a number the adversary chose and ours one never below the loss incurred.

Which trace each recomputed column is of. Its own: the generator refuses to emit a row without a defended recomputation rather than falling back, and the harness counts a defended run's executed steps as the records that received a *token*: a refused record is served no batch, so it is an attempt and not an execution. Where the defended trace contains re-executed indices, its recomputation prices them unamplified, because that is what they cost — so the replay row's two defended columns agree, and the gap between the defended and undefended recomputations on that row *is* the attack. The outcomes differ genuinely. Forking by an undeclared replica is prevented, a record from a replica with no registered identity having no head to extend. Replay is *not* prevented and must not be: the re-executed step touched the data, so the duplicate is charged and logged rather than refused — refusing duplicates would be the natural implementation and a privacy bug. There is no faithfulness row, for the reason Section 3 states: DP-ATTEST does not bind the noise path, and a table claiming otherwise would be the substitution this paper is about.

The defence matrix is a *diagnostic replay*, not the fixed-budget service of Table 6: it feeds each attack trace through the accountant to show the cost that trace requires, without stopping at $\varepsilon = 1$. That is why its defended replay row can report a very large conservative unamplified charge. In the policy-enforcing training run, admission checks the complete prospective vector and refuses before minting any capability that would cross the declared budget. The targeted near-boundary retry regression exercises that distinction directly. A value above one in the diagnostic table is a price for the supplied trace, not evidence that the fixed-budget gate released an over-budget output.

OPENDP, and unforgeable state that cannot cross a restart. OPENDP is the instructive opposite in Table 5 and the reason the table has a *reset* row rather than only a *rewound* one. Its compositor holds the remaining budget behind an FFI pointer into the Rust core, so no caller can forge that state — and none can save it either, leaving a crashed job a fresh compositor with the whole budget again. Unforgeable state that cannot cross a restart is not continuity; it turns a silent rewind into a silent reset, which is the same P5 failure reached from the opposite design decision.

The audit ceiling, stated fully. Section 4.1 reports the ceiling in one clause. In full: a one-run membership audit measures average-case distinguishability, which is governed by the mechanism's Gaussian-DP μ rather than by ε at $\delta = 10^{-5}$, a tail quantity. At $\sigma = 1.5$ and a published 1.95, μ is 0.41, and no audit of this form can certify more than 0.26 *for that mechanism and that executed trace* (Section B). The scope is not a formality: Table 26's own deployment block moves the ceiling from 0.26 at 300 executed steps to 0.87 at 2,400, against the same published 1.95 — the ceiling is a function of (σ, q, T), not of the number on the certificate. Nor is it a statement that every auditor is powerless: it bounds one-run membership audits of this form, and a different instrument with a different estimand is not excluded. Both endpoints of the trend reported in the main text sit under two percent of the published number, which is why the trend and the ceiling have to be read together: the first says unaccounted steps leak more, the second says a relying party cannot use that fact as an instrument.

The two gradient-service comparators, in full. Section 4.3 compresses these into a clause each. *Trusted grad. svc.* holds the same data, mechanism, workload and seeds as DP-ATTEST but differs in *two* features at once: in-memory charging sealed periodically, and no external register to tell a restart whether the state it loaded is the state it left. At a crash every 11 it executes 71 steps against 65 accounted, recomputing to 6.12, where DP-ATTEST executes 38 at 0.78; its 73.9% against our 61.4% is unaccounted steps sold as accuracy, the same trade the evasion row makes with no adversary at all. Two features moving together is exactly why that row cannot carry the comparison on its own. *Durable grad. svc. ($k = 1$)* is the matched test that can: same plane, mechanism, budget, external freshness, workload and seeds, with every release charged durably instead of k reserved ahead. At the harsh schedule it executes 65 steps against 65 accounted, recomputes to 1.00, and reaches 73.1% — matching safety and utility while paying the unamortised anchor cost priced below. The periodic comparator's gap closes at the gentlest schedule in Table 6.

Table 4: Every attack re-run against DP-ATTEST.

	Undefended		DP-ATTEST		
Attack	Rep.	Recomp.	Rep.	Recomp.	Outcome
rollback	1.000	1.061	1.062	1.061	charged
gate evasion	1.000	13.865	1.000	1.000	prevented
fork	0.714	0.885	0.714	0.714	refused
replay	0.727	0.739	2010.134	2010.134	charged
unrecorded steps	1.001	1.167	1.001	1.000	starved

Table 5: Deployed accountants under a rewound composition state, through documented entry points only.

Package	Accountant	Cont. check	Resume	Rep. ε
Opacus 1.6.0	RDPAcct.	none	rewound	$4.08 \to 2.87$
dp_accounting 0.5.1	RdpAcct.	none	rewound	$4.08 \to 2.87$
dp_accounting 0.5.1	PLDAcct.	none	rewound	$3.72 \to 2.58$
prv_accountant 0.2.0	PRVAcct.	none	rewound	$3.73 \to 2.59$
autodp 0.2.3.1	Composition	none	rewound	$4.10 \to 2.87$
diffprivlib 0.6.6	BudgetAcct.	none	rewound	$4.00 \to 2.00$
OpenDP 0.13.0	Sequential	none	reset	$4.00 \to 2.00$

Table 6: DP-SGD on UCI optical digits with the gate in the loop, over 40 seeds.

		Steps		ε		
Crash	Run	Exec.	Acct.	Rep.	Recomp.	Acc. (%)
none	honest	65	65	1.00	1.00	77.8 ± 4.4
	evasion	585	65	1.00	3.01	91.9 ± 1.3
	trusted grad. svc.	65	65	1.00	1.00	76.8 ± 3.9
	durable grad. svc. ($k = 1$)	65	65	1.00	1.00	77.2 ± 3.1
	DP-ATTEST	65	65	1.00	1.00	77.8 ± 4.4
	DP-ATTEST, grad. plane	65	65	1.00	1.00	77.9 ± 4.0
every 11	honest	65	65	1.00	1.00	72.5 ± 5.2
	crash + resume	86	65	1.00	1.14	77.8 ± 4.4
	trusted grad. svc.	71	65	1.00	6.12	73.9 ± 4.4
	durable grad. svc. ($k = 1$)	65	65	1.00	1.00	73.1 ± 4.4
	DP-ATTEST	38	65	1.00	0.78	62.4 ± 7.1
	DP-ATTEST, grad. plane	38	65	1.00	0.78	61.4 ± 6.0
every 23	honest	65	65	1.00	1.00	76.3 ± 4.5
	crash + resume	77	65	1.00	1.08	77.3 ± 3.5
	trusted grad. svc.	74	65	1.00	7.68	76.0 ± 4.2
	durable grad. svc. ($k = 1$)	65	65	1.00	1.00	75.8 ± 3.8
	DP-ATTEST	51	65	1.00	0.89	72.5 ± 5.3
	DP-ATTEST, grad. plane	51	65	1.00	0.89	72.6 ± 5.4

In Table 7, "class. q" and "class. 1" are the service's classifications when it minted each capability; "required 1" is reconstructed independently from repeated sample indices, so it exposes a periodic service that forgot a release. "Durable base" is the surviving prepaid/base-unit count, and forfeiture is unused prepaid capacity observed immediately before recovery. The safe crash rows advance beyond the burned high-water mark and contain no retry surcharge; the old-index/new-query targeted trace exercises that path.

The operating point, as a curve rather than a verdict. Table 8 answers the question the durable $k = 1$ row raised and could not settle on its own. The matched durable service *is* $k = 1$ — same gradient plane, mechanism, budget, workload, external register and seeds — so the two are not rival designs but the ends of one sweep, and the result is the shape of the tradeoff rather than a claim that

Table 7: Raw event semantics for the defended rows of Table 6, as per-seed means over 40 seeds.

Crash	Plane	Crashes	Attempts	Minted	Releases	Distinct	class. q	class. 1	req. 1	durable base	Forfeit	ε (raw)
none	record	0	66	65	65	65	65	0	0	65	0	0.995726
none	durable $k = 1$	0	66	65	65	65	65	0	0	65	0	0.995726
none	gradient	0	66	65	65	65	65	0	0	65	0	0.995726
none	periodic	0	66	65	65	65	65	0	0	65	0	0.995726
every 11	record	3	39	38	38	38	38	0	0	65	27	0.995726
every 11	durable $k = 1$	5	66	65	65	65	65	0	0	65	0	0.995726
every 11	gradient	3	39	38	38	38	38	0	0	65	27	0.995726
every 11	periodic	6	73	72	71	71	72	0	6	65	0	0.995726
every 23	record	2	52	51	51	51	51	0	0	65	14	0.995726
every 23	durable $k = 1$	2	66	65	65	65	65	0	0	65	0	0.995726
every 23	gradient	2	52	51	51	51	51	0	0	65	14	0.995726
every 23	periodic	3	75	74	74	74	74	0	9	65	0	0.995726
every 41	record	1	57	56	56	56	56	0	0	65	9	0.995726
every 41	durable $k = 1$	1	66	65	65	65	65	0	0	65	0	0.995726
every 41	gradient	1	57	56	56	56	56	0	0	65	9	0.995726
every 41	periodic	1	67	66	66	66	66	0	1	65	0	0.995726

Table 8: The commit window k as a curve, from the durable $k = 1$ service upward.

				ms / useful step			
k	Useful	Forfeited	Anchor	register	measured	Acc. (%)	Δ acc. vs k=1
1	65	0.0	66	10.15	2.41 [2.34,2.49]	73.1	—
2	60	5.0	34	5.67	2.13 [2.10,2.15]	70.1	-3.0 [-4.8,-1.2]
5	49	16.0	14	2.86	2.16 [2.14,2.18]	68.4	-4.7 [-6.6,-2.7]
10	38	27.0	8	2.11	2.20 [2.19,2.21]	61.4	-11.6 [-13.8,-9.5]
20	38	27.0	5	1.32	2.20 [2.18,2.23]	60.3	-12.8 [-15.2,-10.4]

either is safe and the other is not. Both are safe; that is what Proposition 1 and Table 34 are for. The runs behind Table 8 crash every 11 steps, over 40 seeds, and every row publishes $\varepsilon = 1.0$.

Four quantities move together and no one of them decides anything. At $k = 1$ the run executes 65 useful steps, forfeits nothing, and performs one register transition per step — 10.15 ms per useful step at the 10.0 ms round trip Table 27 measures. Amortising the transition over a window buys that back: at $k = 20$ the same calculation gives 1.32 ms per step, a factor of 7.7, at the price of 27.0 prepaid units a crash discards unspent and 38 useful steps. Accuracy tracks useful steps down the curve rather than being independent of them — 73.1% at $k = 1$ against 60.3% at $k = 20$, a paired per-seed difference of -12.8 points with a 95% interval of $[-15.2, -10.4]$ — so the window is bought with model quality and not only with forfeited units, which is the part a summary quoting only anchor calls per step leaves out. The interval is what makes the curve readable, and at $k = 2$ it needs care: the paired difference is -3.0 points with interval $[-4.8, -1.2]$. That interval excludes zero: at $k = 2$ the loss is resolved. The clause is generated from the interval, like the equivalence verdicts below, because a hand-written reading goes stale the moment a rerun moves it.

Which column is a calculation and which is an observation. The *register* column is (anchor transitions) $\times$ (round trip)/(useful steps): a measured transition count priced at a selected latency. It is *not* an observed elapsed training time, and it falls by a factor of 7.7 across the sweep because the transition count does. The *measured* column beside it is an observation: the wall-clock of the same runs, accountant and gradient plane and optimiser included, with the register an in-process object carrying no round trip.

What the measured column establishes. It is wall-clock, so it is only as good as the machine was quiet, and it records the load average it ran under — 27 on the 32-core host for this table — with a 95% interval per row. Across the sweep it reads 2.41/2.13/2.16/2.20/2.20 ms per useful step for $k = 1, 2, 5, 10, 20$ (intervals ± 0.01 to ± 0.07), while the calculated register column falls from 10.15 to 1.32 ms, a factor of 7.7. The measured column is flat to within 13%, so shrinking the register's transition count did not make the run faster. With the register in process it contributes almost nothing to elapsed time, which is precisely why the two columns must not be added. Measurements on this machine at other loads order its rows differently, so no trend in k should be read from this column.

The scale is the point, and it is why the two columns must not be added. A register on a low-latency path leaves the measured column as essentially the whole cost, and k is then a forfeiture decision rather than a latency one; a register at 10.0 ms makes the calculated column dominate, and k becomes the latency decision that column prices. Which regime a deployment is in is a fact about its register, not about this table. Figure 3(b) and Table 27 price a real socket separately, on a separate harness.

Forfeiture is the constant, and it is not the envelope. Every restart discards the outstanding reservation — clean, torn or rolled back alike — so "recovery adds no charge" never means "recovery costs nothing". The *Forfeited* column is that cost, and it is what rises with k; the accounting charge a recovery adds is separate, is zero when the register's state can be produced, and is an exact reconstruction otherwise (Section A.2). The recommendation a deployer can act on is therefore conditional and is the one Section C.3 states: size k from the register's round trip and the crash rate, take $k = 1$ where the register is on a low-latency path, and do not read fewer anchor calls as better in itself.

The equivalence tests, and the preemption bill. Table 6 is the table Section 4.3 quotes from. In Table 6, every row publishes its *reported* column, and *recomputed* is reconstructed independently from the trace, pricing any observed reuse of a hidden sample without amplification.*The decision rule.* The test is a paired two one-sided t-test (TOST) on per-seed accuracy differences, margin ± 2 points declared before the runs, at $\alpha = 0.025$; equivalence is declared iff $\max(p_-, p_+) < \alpha$, which is identically containment of the 95% interval inside the margin, since TOST at α *is* containment of the $100(1 - 2\alpha)\%$ interval. Stating the level matters because the two conventions can disagree: a p between 0.025 and 0.05 is equivalence at the conventional level and not at ours. The generator asserts that the two criteria agree before emitting either, so they cannot disagree in print. The level is the conservative choice, which is the direction a privacy claim should err in.

Both verdicts below, and the clauses that introduce them, are generated from the same numbers, because a hand-written interpretation goes stale the moment a rerun moves a result; the generator emits each verdict together with its "does/does not" clause.

Over 40 seeds the record plane against honest training is +0.00 points, $[+0.00, +0.00]$: the test supports equivalence at the declared margin (established, $p = 0.000$ against $\alpha = 0.025$). The gradient plane against the record plane is +0.14 points, $[-1.02, +1.31]$: the test supports it (established, $p = 0.001$). Both come from runs that divide by the public qN (row 15 of Section A.3): the per-seed rows, the declared margin and the paired summaries are generated together in `real_training.json`, whose hash the manifest records. A *not established* verdict, wherever one occurs, is neither a demonstrated accuracy loss nor a refutation of equivalence — it means this margin and level were not met at this seed count, so the comparison is inconclusive rather than favourable. Where a contrast reports zero observed variance (yes / no for the two rows) the interval collapses to a point and the declared rule is containment of that point, not a t statistic; the generator flags the case rather than printing a degenerate p as though it were an ordinary test. The lower blocks of Table 6 are the defence's bill: under preemption the reservation forfeits its unspent tail, so at a crash every 41 steps DP-ATTEST returns 73.4% where the unprotected run returns more because it is over budget rather than better (the executed trace recomputes to $\varepsilon = 1.04$ against 1.0 reported).

Where the overhead goes. The end-to-end measurement of Figure 3(b) excludes nothing: omitting the anchor round trip or the plane would measure hashing rather than the system. Index derivation is not critical-path work, because the beacon fixes the schedule at step 0, at 130 µs per step of background compute. The $1/k$ curve falls until it reaches the floor set by per-step MAC and hashing work, which the two low-latency curves hit by $k \approx 100$; anchor latency sets where that floor lies.

Where a fix would go. The location of the missing check in Table 5 cuts against the libraries that hold nothing. Opacus holds composition state and restores it unconditionally, so the rewind happens inside library code and could be caught there; for the four stateless accountants P4 lives outside the library entirely, and no change to the library could catch it.

Multi-replica semantics. The interleaving enumeration of Section 4.3 evaluates the invariant after each event as *what the accounting would be if the accountant died now*, which is the only formulation under which a crash is not a special case. It applies that formulation to both the durable count and the durable Rényi vector, rather than inferring heterogeneous costs from counts. Multi-replica

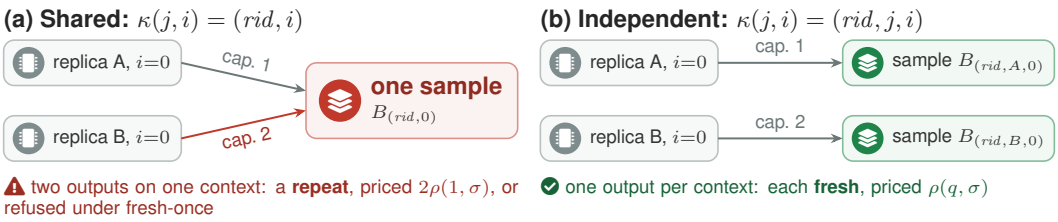


Figure 5: Shared and independent sample contexts for two replicas at one index.

Table 9: The sample-context rule, run against the implementation.

Case	Mode	Disp.	s	Counts	Durable	Required	Holds	Non-int.
two replicas, one context (k=2)	shared	CD	2	2/2/2/1	2.0000000	2.0000000	yes	yes
same, per-replica rule	shared	CC	0	2/2/2/1	0.0000034	2.0000000	**no**	yes
both held, redeemed in reverse	shared	CD	2	2/2/2/1	2.0000034	2.0000000	yes	yes
same, independent	ind.	CC	0	2/2/2/2	0.0000069	0.0000034	yes	yes
clean restart between uses	shared	CD	2	2/2/2/1	2.0000103	2.0000000	yes	yes
stale restart between uses	shared	CCCCDD	4	6/6/6/4	4.0000034	4.0000034	yes	no
three replicas, one context	shared	CDD	3	3/3/3/1	3.0000086	3.0000000	yes	yes
independent control (k=2)	ind.	CC	0	2/2/2/2	0.0000034	0.0000034	yes	yes

semantics are checked the same way: concurrent declared replicas both charged, an undeclared replica refused, a same-head race serialised, a mid-run join charged once registered (and, under shared contexts, charged as a repeat on the contexts it re-reads), the byte-level shared-context check, and every ordering of the three crash events safe; the sample-context cases are in Table 9.

What the primitives are worth, and the cost split. Section B prices each variant exactly. Deliberate rollback is the same primitive as a gate that never closes; forking needs no restart at all, only that the accountant be consulted by several processes each believing it is alone, so every replica gets an honest answer to the question it asked and the error is in the question; and replay is the defensive result, understating by a negligible 1.02× while an accountant holding durable indices sees 1,000 arrive twice — direct evidence of a rollback, circular unless broken by the register. Section B.3 adds two levers that bound what a correct accountant is worth alone, neither new (Chua et al., 2024; Cebere et al., 2026). Table 29 splits the data plane's cost by party; the largest single item, the auditor's multiproof verification, is off the training path entirely.

Sample contexts and replicas, run against the code. Three identities are kept apart throughout, because conflating any two of them is a natural defect (Section A.3): the *replica* j (whose chain and whose contiguous clock), the *sample context* $\kappa(j,i)$ (which hidden sample a release reads), and the *capability* (which noise draw it carries — the plane seeds noise from it, so one capability is one output). The checker's oracle holds the plane's own ordered release log, outside anything a rollback of the accountant reaches, and prices what the plane actually released by the strict rule: a context carrying T distinct outputs owes $\rho(q,\sigma)$ if $T = 1$ and $T\rho(1,\sigma)$ otherwise. An oracle keyed on (replica, index), or one that priced amplified-first, would not check the rule Proposition 1 states, and the first cannot see two replicas reading one sample.

Figure 5 draws the two modes for two replicas at one local index. When replicas *share* contexts, both read the same hidden sample, and because the plane seeds noise from the capability each receives its own full noised output: the second is a repeat on that context, priced unamplified with the first upgraded ($2\rho(1,\sigma)$ in all), or refused under a fresh-once registration — which is why fresh-once requires independent replicas. When replicas are *independent*, the replica is part of the context, the two samples are independent, and each release is fresh at $\rho(q,\sigma)$.

The traces of Table 9 are `shared_context_traces` in `state_machine.py`. Dispositions: C charged, D duplicate (a repeat on an already-authorised context). Counts are minted capabilities / releases / distinct outputs / distinct sample contexts. *Durable* is the order-two Rényi cost a crash at the end would recover; *required* is the strict price of what the plane actually released, per context.

Table 10: The mixed-replica trace as a per-event ledger.

Event	Disp.	Context	Minted	Rel.	Outputs	Contexts	Accounted	Required
A@0 (u0)	charged	(A,0)	1	1	1	1	0.0000069	0.0000017
B@0 (u0)	charged	(B,0)	2	2	2	2	0.0000069	0.0000034
B@1 (u0)	charged	(B,1)	3	3	3	3	0.0000069	0.0000052
B@0 (new-query)	duplicate	(B,0)	4	4	4	3	2.0000103	2.0000034

The last column is the privacy corollary's hypothesis over the plane's release order, not a property the certificate carries.

The two-event case, against the code. Shared contexts, $k = 2$: A's release at index 0 reserves two units and spends one, and B's release at index 0 reads the same hidden sample. Under the context rule (line 12) B's is a repeat owing $m = 2$ surcharge units, and the durable order-two cost reaches 2.0000000, the strict requirement $2\rho_2(1, \sigma)$ exactly. A per-replica rule, with both tests keyed by replica, runs beside it as a control: it calls B's release fresh and ends at 3.4×10^{-6}, which is not only below the strict charge but below the exact order-two divergence of two independently noised releases on one hidden sample in the scalar construction of Section B.1, 6.4×10^{-6} at $q = 10^{-3}$, $\sigma = 1$ (two fresh charges would be 3.4×10^{-6}). The checker reports that control's violation (Table 9, second row); the count invariant survives and the price does not, which is why the count alone is not the property.

The cases a served-output rule would miss. The repeat test reads *authorisations*, because that is where the charge is made: with both capabilities minted and neither redeemed, B's is already a repeat. Redeemed in reverse order, both outputs leave, B's first, and the context carries two — priced as two. B's surcharge was sealed by a commit before B's capability was minted, so A's capability now predates the register's generation; the gradient plane does not refuse it for that, because its output is a PRF of the capability and a re-presentation is post-processing. (Applying the record plane's generation floor here would expire A's capability: safe, since A was charged, but a needless loss of a paid release.) Recovery between two uses of one context, clean or from a stale blob, leaves the second use a repeat in both cases: the exact path through the sealed b_r, the reconstructing path through the burn to $B^\star$. Three replicas on one context pay the upgrade once (s ends at 3, not four).

The mixed-replica ownership trace. This trace exists for a different defect — sealing b_r for the requesting replica only (Section A.2) — and runs with *independent* replicas, $k = 4$: A mints the window and releases at index 0; B releases at indices 0 and 1 from the pool with no commit of its own; a clean restart; B re-executes index 0 at a new query. That is 4 minted capabilities, 4 releases, 4 distinct outputs and 3 sample contexts — $(A, 0)$ once, $(B, 0)$ twice, $(B, 1)$ once — so the strict requirement is $2\rho_2(1, \sigma) + 2\rho_2(q, \sigma) = 2.000003$ (Table 10). The implementation accounts 2.000010; the requester-only control, whose restart returns $D.\mathrm{high}[B] = -1$ and prices B's re-execution as fresh, accounts 0.000014 and fails. An oracle that groups redemptions by (index, query) — the one-object-per-index convention that the byte-level check below disproves — misprices this trace, which runs in independent mode (row 13 of Section A.3). Its release order is interleaved (context $(B, 0)$ is re-entered after $(B, 1)$), so this trace is evidence for the bookkeeping proposition and not an instance of the privacy corollary's hypothesis.

Two details of the construction matter, because getting either wrong hides the defect. k must be large enough that B's releases come from A's pool: if B mints a reservation of its own it seals b_B and the control passes. And the last event must be a *re-execution*, not a resumption — a rolled-back high-water mark also rolls back the expected next index, so a resumed index is refused by the contiguity rule of condition (vi). Contiguity is a second, independent guard, and it covers exactly the case re-execution does not.

What the plane releases on a shared context. `multi_replica.shared_context_noise` compares the bytes: two replicas on one context receive independently noised full gradients (maximum coordinate difference 0.011), not fragments of one draw, and the accountant charges them as a fresh release and a repeat on one context. Charging the pair once, as one partitioned application, would be unsound: the plane never slices a draw, so as built the shared configuration buys no accounting saving over independent replicas. A plane that noised once and served disjoint fragments would restore the single charge; we have not built one.

Availability, asserted rather than assumed. Every checker in this paper shares one blind spot: an accounting checker can satisfy *every* safety assertion while the useful service has stopped, because REFUSE is always safe. A shortened final window that lowers a sealed bound has exactly that shape (row 8 of Section A.3): nothing is released uncharged, and the run cannot progress. `mixed_replica_traces.availability` therefore records, for three ordinary situations, whether the run *progressed* (Table 11).

Table 11: Whether the run progressed in three ordinary situations.

Case	Outcome	Progressed
ordinary completion (8 affordable steps)	8 served, no latch	yes
clean recovery (restart, then resume)	next index charged	yes
shortened final window (5 units, $k = 4$)	5 served, no latch	yes

The third case shrinks the reservation, and its trace prints the frontier before each event, the frontier the transition proposes, and the return status. With two replicas and five affordable units, $A0, B0, A1, B1$ seal $b = 3$ and the register moves to $(3, 4, 0)$; the fifth step needs a *one*-unit window, and because line 17 takes $b_r \leftarrow \max(b_r,\, D.\mathrm{high}[r] + k')$ it proposes $(3, 5, 0)$ rather than $(2, 5, 0)$ and returns CHARGED. The reservation therefore retains old bounds when k' shrinks, and this is a field in the published trace rather than a claim here.

A.5 THREATS TO VALIDITY AND RELATED WORK, IN FULL

Threats to validity. Each threat follows from a specific choice in the study design, and they are grouped as Section 5 groups them: construct, internal, external, and conclusion validity. *Conclusion validity: seed counts.* Seed counts were set for interval width rather than convenience: 40 for the paired accuracy comparisons, which is what a ± 2-point equivalence margin needs, and 15 for the audits, where the reported quantity is a lower bound rather than a difference.

Construct validity: the hardware gap. Nothing was produced on confidential hardware. This does not weaken the attacks, each being executed by the party that supplies the image through interfaces the guest controls, and neither platform exposes a sealing freshness primitive. It weakens two claims. Figure 3(b) measures a commodity CPU against a CPU step time, so a confidential-GPU deployment should recompute the ratio, and the direction is favourable — our own step-time sweep spans $452\times$. And the register's *unrewindability* is inherited from whatever provides it: Section 4.3 attacks the protocol around it, not the counter's physics, and Section C.5 says which platform facility suffices against which adversary.

Construct validity: the commitment's hiding property is computational. The published dataset root is hiding under a sealed per-run salt, which is what makes the ordinary add/remove pair admissible (Section B.1); the resulting statement is therefore computational (ε, δ)-DP rather than information-theoretic. DP-ATTEST is already in that regime through the mechanism's PRF noise and through the sampler, whose seeds and batch commitments are keyed by one secret β whose hash is published — assumption (S*), which we justify in the random-oracle model and do not reduce further (Section B.1). So this adds a reduction and not a new kind of assumption — but an information-theoretic claim about the deployed transcript is not available here, and we do not make one.

Internal validity: what the library study shows. No library studied is shown defective, and no deployment is shown exposed: we surveyed none and probed no hosted service, so Table 5 reports an absent check in components whose threat model never included a rewinding caller, not a census of confidential-training systems. The leading adversary's realism rests on structure — these are ordinary dependencies outside every measurement the platform takes — and on Table 25, where the same loss occurs with no adversary at all.

Internal validity: the audits are lower bounds. Figure 6 and Table 23 use orthogonal canaries on a linear model, the configuration in which an audit is most powerful, and report bounds below the true ε. Figure 6 evidences a trend and a floor; Table 23 uses the instrument the other way, to establish that a quantity is nonzero rather than how large, which is the one thing a lower bound does without qualification. The systems argument does not rest on it: the threat model grants the platform the checkpoint bytes, so the discarded iterate is an available output whether or not this particular auditor

Table 12: What (q, σ) moves: the budget's step count, the attacks' recomputed ε, and the share of it crashes forfeit.

			Recomputed ε		Forfeited, 10 crashes	
q	σ	$T^\star$	omission	rewind	$k = 10$	$k = 100$
0.001	1	27,399	1.17	1.32	0.2%	1.8%
0.001	1.5	108,025	1.25	1.46	<0.1%	0.5%
0.001	2	214,069	1.25	1.46	<0.1%	0.2%
0.001	3	518,946	1.25	1.46	<0.1%	0.1%
0.01	1	2	1.03	1.05	100%	100%
0.01	1.5	975	1.23	1.43	4.6%	50.8%
0.01	2	2,045	1.24	1.45	2.2%	24.2%
0.01	3	5,098	1.24	1.46	0.9%	9.7%
0.05	1.5	18	1.13	1.24	100%	100%
0.05	2	65	1.21	1.39	69.2%	100%
0.05	3	188	1.23	1.43	23.9%	100%

resolves it. Table 6 is a real dataset but a small one, and only the evasion gap and the gradient-plane equivalence should be read as results.

External validity: one point in parameter space. Every analytic number comes from one (q, σ, δ), fixed before the attacks were run; the qualitative structure follows from composition but the specific multiples do not transfer. The two parameters we sweep are the ones a reader would most suspect of selection: the published budget (Figure 3(a)), where the attack's value falls to zero, and the commit interval (Table 34), where the write-ahead ratio is flat at 1.00 while the design it replaced spans 40.20–40.98×. The workload is one small task on CPU, which bounds what the cost ratio transfers to.

A third parameter is swept in Table 12, because a reader should not have to take on trust which conclusions travel. They sort cleanly. The *systems* invariant is parameter-free: the proof of Proposition 1 and the checker's enumeration never use the numeric value of (q, σ) — the prices are symbols that add — so accounted $\geq$ executed, the dominance of R, and one output per context under fresh-once hold at every cell. The *quantitative* results all move through one number, the step count $T^\star$ the budget buys, which spans 2 to 518,946 over the grid. Where the budget buys a real run (at least 50 steps) the omission attack's recomputed ε stays within 1.17–1.25 and a rewind's within 1.32–1.46, against a report of 1.0: the multiplier is stable because composition grows roughly with the square root of the count whatever the per-step price. What a fixed number of crashes forfeits is not stable, because it is a count of units set against $T^\star$: 10 crashes forfeit 69% of the training run's 65-step budget at $k = 10$, but 1.8% of the analytic regime's at $k = 100$. The operational rule follows: size k against $T^\star$ and the crash rate, not against the register's latency alone. Table 12 is closed-form accounting at $\varepsilon = 1$, $\delta = 10^{-5}$; nothing is trained or timed. $T^\star$ is the step count the budget buys. The omission and rewind columns recompute ε when $1.5T^\star$ and $2T^\star$ steps run behind a report of 1, the ratios of Table 21 and of one rewind cycle. The forfeiture columns are the expected share of the budget that 10 crashes forfeit under write-ahead, $(k - 1)/2$ units each. Cells where the budget buys no step are omitted.

External validity: recovery is exact, and what it costs is a stronger register. Recovery reconstructs R and the possible-index bound from the register's frontier rather than bounding them (Section A.2), so a rolled-back run neither under-charges nor pays a worst case — which is a better property than the per-generation envelope this replaced, and a *narrower* platform assumption is what buys it. The register must hold a generation, a digest and three nondecreasing integers, answer them as one authenticated snapshot bound to the request that asked for it, and refuse a regressing advance (Section A.2); the binding is built on both register paths and tested against a replaying host. That is more than a bare monotonic counter, it is less than a general trusted store, and a platform supplying only the counter keeps the guarantee and loses the ability to continue past a rollback at all. We state it as a requirement rather than an implementation detail because it is the assumption a deployer has to check.

Scope: invisible is not undetectable. The attacks are invisible to the accountant and to the report, not to an operator with complete and trustworthy infrastructure logs. Such logs are neither attested nor part of what a relying party receives, which is why we do not count them.

Related work: placing the systems. Our results do not contradict the lines Section 5 surveys; they locate them (Table 13). Proof systems certify P1–P3 far more strongly than we do — VERIDP binds a noise path we do not implement at all — but only for a *declared* step count, and Table 4's evasion row is what a declared count is worth. The right reading is compositional: DP-ATTEST supplies history and completeness, a proof system supplies mechanism fidelity, and a model-level guarantee needs both. State-continuity mechanisms make P5 durable, which our replay result confirms is necessary and our omission result shows is not sufficient; budget-gated access works because its gate sits in trusted infrastructure, and transplanting it under an untrusted host is what forces the register, the token and the write-ahead window. The paragraphs below give the long form of Section 5, one line of work at a time, ending with the transparency-log anchor.

The careless-implementer line, attributed individually. Section 5 compresses this line into one sentence; each contribution is owned by its own paper. Grey-box auditing of declared against measured sensitivity has found violations across deployed libraries (Cebere et al., 2026), and owns the machinery Section B.3 borrows — the difference is only in the adversary. The shuffled-loading versus Poisson-accounting discrepancy is owned by prior measurement (Chua et al., 2024), and floating-point attacks by earlier work (Mironov, 2012; Jin et al., 2022). Reverse-engineering of a proprietary DP framework (Chourasia et al., 2026) models the vendor as honest and opaque rather than as rewindable. Our audits apply the one-run form (Steinke et al., 2023) of an active empirical-ε line (Jagielski et al., 2020; Nasr et al., 2023; Annamalai & De Cristofaro, 2024; González et al., 2025). And the Gaussian Pancake mechanism (Sun & He, 2026) owns the covert weakened-noise contribution outright: its adversary need not survive verification, being undetectable by indistinguishability rather than because a verifier checked the wrong thing, where ours submits to verification and passes.

What does not carry over from a query service on SGX. Section 5 names the substantive difference; there are three. First, a per-query continuity round trip is affordable when a query is the unit of work and is not when a gradient step is, which is why Figure 3(b) and Table 8 are amortisation results rather than overhead measurements. Second, SGX's monotonic counter was withdrawn, and confidential VMs expose no equivalent: the state a guest can persist is host-managed, which is what forces the external register of Section 3.1 and makes its availability a deployment question (Section C.5). Third, and substantively, in an interactive service the TEE that answers a query *is* the TEE that charges for it, so declining to charge means declining to answer. In training they are separate components inside one VM, so an adversary controlling the trainer need not roll the accountant back when it can simply decline to call it — which is the omission attack of Section 4.2, and the reason the design gates the data rather than auditing the ledger.

Access decisions inside a TEE. Putting an access decision inside a TEE is established, from policy-enforcing object stores and hardened key-value stores (Krahn et al., 2018; Bailleu et al., 2019) to end-to-end encrypted analytics (Zheng et al., 2017; Hunt et al., 2016) and vaults releasing a key against a report matching a declared measurement (Microsoft, 2023); our token is a capability in the sense of macaroons (Birgisson et al., 2014). What none supplies is the *caveat*: those predicates range over identity, policy and measurement — who is asking and what code is running — where ours ranges over the asker's history, and separating the thousandth step of a run from a replay of the first is not something a measurement can decide.

Proving a budget rather than its preconditions. Zero-knowledge certification of a declared (ε, δ) (Shamsabadi et al., 2024) binds a final budget, not the preconditions under which it is the right number — the same boundary Section 5 draws around VERIDP, reached from the other end.

What a proof chain quantifies over. Section 5 states that VERIDP's proof certifies the composition arithmetic for a *declared* count. The reconciliation is as follows: the iteration count is a public parameter of the statement proved, so no statement of that shape can certify that the accountant's state survived a restart, a fork or a replay — those are properties of a long-lived service that chooses T, not of the arithmetic over a T already fixed. Nor could a proof replace the anchor: two honest proofs

Table 13: What each system binds. DP-ATTEST supplies P4–P5 and composes with a mechanism-fidelity system for P1–P3.

	P1–P3 mechanism	P4–P5 history	Untrusted host	Rate
VERIDP (Abdolmaleki et al., 2026)	✓	declared T	✓	per step
ELEPHANTDP (Jin et al., 2024)	—	ledger	✓	per query
SAGE (Lécuyer et al., 2019)	—	✓	—	per pipeline
State continuity	—	ledger	✓	n/a
DP-ATTEST	—	✓	✓	per step

of one statement, one from the live run and one from a checkpoint restored beneath it, are equally valid, so freshness must come from something monotone the prover does not hold (Section C.4).

Table 13 places the systems Section 5 concedes to. DP-ATTEST is the only entry binding the execution history under an untrusted host at SGD rate, and the only one with a dash in the first column — which is the composition drawn above.

The rollback line, and the nearby systems. The trusted-computing work on rollback resistance we consume rather than extend: MEMOIR (Parno et al., 2011), ICE (Strackx et al., 2014), ARIADNE (Strackx & Piessens, 2016), ROTE (Matetic et al., 2017), NARRATOR (Niu et al., 2022) and NIMBLE (Angel et al., 2023) each build the register Section 2 assumes, and CACHEWARP (Zhang et al., 2024) is a reminder that a confidential VM's own state can be rolled back by the host it runs on. What this paper adds is not another such mechanism but the observation that protecting the ledger does not protect the guarantee when the component that reads the data and the component that charges for it are separate. Nearby, an ephemeral server-side TEE for maliciously secure DP federated learning already flags forking and state management (Takagi & Hasegawa, 2026), and attestation has been proposed for certifying what a model release advertises (Duddu et al., 2025) — which is the composition this paper argues does not hold for ε.

The production system Section 1 opens with answers freshness differently again. A public transparency log (Van Overveldt & Ramage, 2025; Eichner et al., 2024) is an anchor with a different profile rather than no anchor — stronger against a colluding operator, needing no per-guest hardware, and publishing the cadence of every run. Its budget is charged per device per round where an SGD accountant composes 10^3–10^5 times, so a log would have to absorb the object Figure 3(b) amortises; a deployment that would rather publish than provision a counter can substitute one without changing anything else here.

B PROOFS AND ANALYTIC ATTACKS

This appendix proves Proposition 1 and its privacy corollary, and then gives the exact consequences of the accounting for each variant of the attack, computed by the accountant of Section 3.2 and summarised in Section 4.1; Section B.3 does the same for the sampler and the clipping site. Parameters throughout: $q = 0.001$, $\sigma = 1.0$, $\delta = 10^{-5}$, claimed budget $\varepsilon = 1.0$, checkpoint every 500 steps.

B.1 PROOFS OF PROPOSITION 1 AND ITS PRIVACY COROLLARIES

Fix the primary configuration of Section 3.2. A *capability* is $c = (rid, j, i, c_i, u_i, a)$ carrying $\mathrm{MAC}_{key}(c)$ under the key the accountant shares with the plane alone. The accountant's sealed state is $\Sigma = (R, n, s, h, D, U, a, w, b)$, where w is the reservation charged but not yet spent — a *run-level pool* any registered replica may spend — b_r bounds the indices that reservation could release *on replica* r, and it is sealed for every registered r; s counts the unamplified surcharge units charged and U the *sample contexts* whose first release has already been upgraded (Algorithm 1). The sample context of index i on replica j is $\kappa(j, i) = (rid, i)$ when replicas share contexts and (rid, j, i) when they are independent; it names the hidden sample a release reads, as the capability names the noise draw. The *strict price* of a distinct released object o is $\rho(o) = \rho(q, \sigma)$ if o is the only distinct output on its context and $\rho(1, \sigma)$ otherwise, so a context carrying T outputs costs $\rho(q, \sigma)$ or $T\rho(1, \sigma)$ in total. Let the reachable states be those produced by any finite interleaving of: AUTHORISE; a *presentation*

of any capability the adversary holds to the plane; an accountant crash *followed by* RESTART, which may reload an equal-generation blob (a clean restart) or a stale one (a rollback); a plane crash; a rollback of the sealed store; and a rollback of every byte of plane state, its redemption set and its freshness floor included. The register is the only state not on that list, by assumption (i).

Write A for the accounting that survives a crash at the current instant — the accounted count Algorithm 2 would come back with — and M for the number of distinct objects the plane has put across its boundary. Which blob recovery ends up holding is part of the statement: RESTART loads the state the register's lineage digest names when the platform serves it, and otherwise the newest blob the platform offers, reconstructing from the register's accounting frontier $F = (B, n, s)$ and refusing when there is none.

Proof. Step 0: the frontier is monotone, outside the rollback, and exact. Three statements, and the whole of what recovery needs. *Monotone:* $\max_j b_j$ never falls, because b_j is only ever raised; n and s are counters that only increase; and by (i) the register refuses an advance that would lower any of them. *Outside the rollback:* F lives in the register, which the reachable-state model does not permit the adversary to touch, and it is written *with* the compare-and-swap, so it records the state of the last successful commit whether or not the seal that followed it landed. *Exact:* by (v) every unit costs $\rho(q, \sigma)$ and every surcharge costs the same Δ, so

$$R \;=\; n\,\rho(q,\sigma) \;+\; s\,\Delta$$

holds after every transition, and (n, s) therefore determines R. Recovery restores B, n and s, hence $D.\mathrm{high}$ and R, to their values at the last commit.

Why a per-generation bound does not suffice. A natural argument bounds a lost generation — R rises by at most $k\rho(q,\sigma) + \Delta$, b_j by at most k — and composes the bound over the generation gap. The second half is false, and the failure is instructive rather than arithmetic: recovery itself raises $D.\mathrm{high}$, and the next reservation sets $b_j \leftarrow D.\mathrm{high}[j] + k'$ *from the raised value*, so a generation that follows a recovery can move b_j by far more than a window. Two surcharge-only commits hold b_j still while the generation counter runs; restoring the older blob then jumps b_j by $2k$ in one generation, and a later restore of an intermediate blob burns short of what the intervening window released. Nothing linear in the gap survives RESTART → AUTHORISE → RESTART, which is why Step 0 is a statement about a quantity the adversary cannot move rather than a bound on what it can do.

Step 1: A never decreases, and neither does its weighted counterpart. A rollback of the sealed store lowers $\Sigma.n$ and leaves the register holding F, so recovery restores exactly what the rollback removed; a register that is monotone and per tenant (i) cannot be lowered, stalled below a value it has returned, or advanced on another holder's behalf, so no schedule lowers A. A sealed generation *ahead* of the register, or a blob whose n or s exceeds the frontier's, can only mean the register itself was rewound, and RESTART refuses — vacuously safe. A generation the register names but no blob realises — a commit torn between its advance and its seal — is reconstructed from F like any other, because F was written by that generation's advance; the digest check never converts a real transition into a free one. And a register that cannot carry F at all leaves nothing to reconstruct from, so recovery from a state it does not name refuses (Algorithm 2, line 11). *The construction therefore requires a register with a payload*, and Section C.5 says which platform facilities have one; a counter-only register keeps the guarantee and loses the ability to continue past a rollback at all.

Step 2: distinct capabilities are bounded by A. Only AUTHORISE mints, the plane never mints, and key is held by those two alone, so a capability the plane accepts is one the accountant minted. We exhibit an injection from minted capabilities to durable charged units. Track the running accounted count n and the outstanding reservation w. Three transitions touch them: a reservation adds k' to *both* n and w and seals (Algorithm 1, line 17); a mint decrements w by one and does nothing else to n (line 27); and RESTART sets $w \leftarrow 0$ unconditionally before charging any rollback gap (Algorithm 2, line 9). Hence every mint consumes a distinct unit of n that some earlier reservation both charged and sealed, and no unit is ever spent twice: a mint spends a unit only while $w > 0$, and the only way to raise w after it reaches zero is a fresh reservation, which first adds the same amount to the durable n. This is exactly what closes **R3's restore schedule**. Suppose the adversary seals a reservation at generation a, mints a capability bound to u_1, redeems it, then restores accountant state — whether to the *same* generation a (a clean reload) or an earlier one (a rollback). RESTART sets $w = 0$ on either path, so a subsequent AUTHORISE finds $w = 0$ and must reserve again before it can mint a capability bound to a different u_2; that reservation charges n and, through COMMIT's

compare-and-swap, advances the anchor. The pre-restore reserved tail is counted in the sealed n but is no longer spendable — it is forfeited, not reused — so the second capability consumes a new durable unit rather than the old one. Every distinct minted capability therefore maps to a distinct unit counted in n at mint time, and by Step 1 still counted in A now. Hence $|\mathcal{C}| \le A$ for the set $\mathcal{C}$ of distinct capabilities ever minted.

The same recovery also preserves the information needed to price a reused sample. A release is priced by its sample context: line 12 asks whether $\kappa(j, i)$ has been *authorised* before, and because fresh indices form a contiguous per-replica clock (vi) that is exactly $i \le \max_{r \in G(j)} D.\mathrm{high}[r]$ — the replica's own mark when replicas are independent, the maximum over every replica when they share contexts. Authorised, not redeemed: a capability minted and never presented already counts, so no order of redemptions — delayed, reversed, or never — changes a price, and a capability that is never released only over-charges. The set of indices any sealed reservation could have released on replica r is an interval ending at b_r, and $B = \max_r b_r$ is therefore the largest index the run could have touched at the last commit. Recovery advances every $D.\mathrm{high}[r]$ through $\max(b_r, B)$ before w is discarded (line 12), so every context released before the crash tests as a repeat afterwards, in either mode: through r's own mark, which b_r bounds, or through the maximum, which B bounds. b_r covers the window the loaded blob itself sealed — the indices whose first use can postdate that blob's last history update — and B covers everything sealed after it, which a stale blob cannot name at all. Because B is read from the register rather than inferred from the gap, this holds however many recoveries and reservations the schedule interleaves; Step 0's monotonicity is the whole argument, and there is nothing to compose. If a burnt index is offered under a different query it receives the unamplified price and that surcharge is sealed before minting; if the trainer instead resumes above the burnt range, the ambiguous region is simply skipped. So losing the latest duplicate history, losing a whole later window, or losing a window opened *by an earlier recovery*, cannot turn a reused hidden sample into a fresh-subsample charge.

Step 3: the released object is a function of the capability. On the record plane the index set is derived from $\mathrm{KDF}(\beta, \kappa(j, i))$, fixed before the run and independent of every piece of plane state, and the leaves are the committed corpus; the plane refuses any c_i that is not the beacon's. On the gradient plane the per-example gradients are taken at the query u_i named in c — a presentation at any other query is refused, not recomputed, because u_i is inside the MAC (iii) — and clipping and summation are deterministic, so the only remaining freedom is the noise, which is $\mathrm{PRF}(\mathrm{seed}, c)$ under a seed sealed to the plane's measurement (iv). In both cases the map $c \mapsto$ output is a function, and by sole custody (ii) nothing leaves the boundary except through it. Hence $M \le |\mathcal{C}|$.

Step 4: the durable vector dominates the released costs. Counting is not enough once releases have different prices: a re-execution is charged $\rho(1, \sigma)$ while a reservation prepays $\rho(q, \sigma)$ per unit, so a schedule can preserve $A \ge M$ while losing the difference. Let $\mathcal{O}$ be the distinct released objects and $\rho(o)$ the accounted cost of o's mechanism. Each o is the image of exactly one capability c (Step 3), and each such c consumed exactly one reserved unit (Step 2). That unit contributed $\rho(q, \sigma)$ to the durable R when its window was sealed. If $\rho(o) > \rho(q, \sigma)$ — the re-execution case — conditions (v)–(vi) add the difference to R and *seal* it before c is minted, so R already dominates $\rho(o)$ coordinatewise at the moment o can first exist. On a context carrying $T > 1$ outputs the first is charged $\rho(q, \sigma)$ and the first *repeat* $m = 2$ surcharge units — its own and the upgrade of the first — then one per further repeat, a total of $T\rho(1, \sigma) = \sum_{o \text{ on the context}} \rho(o)$, which Section B.1 shows dominates the reused-inclusion mechanism at every order. The upgrade is recorded in U by context and joins it before the seal that pays it, so it is owed once, whichever replica repeats first; a stale recovery that loses the record pays it again, the permitted direction.

That settles every state in which R is the vector the run has been accumulating. A restore replaces it, so the claim has to be re-established rather than inherited. Index the commits $c_1, c_2, \ldots$ and write Σ_g for the state c_g sealed and $\mathcal{O}_g$ for the objects minted in (c_g, c_{g+1}). The mints in that interval draw on the reservation c_g sealed, and a repeated context among them would need a surcharge commit, which *ends* the interval — so write-ahead gives $R(\Sigma_g) \succeq \rho(\mathcal{O}_0 \cup \cdots \cup \mathcal{O}_g)$ for every g. Let G be the generation the register reads. Recovery does not have to reconstruct the *path* from a restored Σ_{g_0} to Σ_G; by Step 0 it restores (n, s) to their values at c_G, and the same two counters determine $R(\Sigma_G)$ exactly. So recovery leaves $R = R(\Sigma_G) \succeq \rho(\mathcal{O}_0 \cup \cdots \cup \mathcal{O}_G)$, which is every object released before c_{G+1} — and the only objects released after it are those the reservation c_G sealed already prepaid. Since R never decreases (Step 1) and every o's cost was dominated when it was released,

$R \succeq \sum_o \rho(o)$ at every later state, restores included, and the argument does not grow with the number of restores because it never composes over them.

Rows 2, 3 and 5–8 of Section A.3 are weaker rules that fail here, each because it reasons about the latest sealed state while the adversary moves the boundary. All are distinct from the replay defect (Section A.4), whose capabilities were legitimately different and whose outputs legitimately new; what was lost there was their price, not their number.

Condition (vii) fixes the boundary of "reachable." A clone made from sealed state must bootstrap through the register, where only one lineage wins the compare-and-swap. A byte-for-byte clone of *live private memory* after a window has been reserved would copy w itself and is not ruled out by that register transition; preventing it requires the deployment's single-live- instance lease, or the non-amortised $k = 1$ mode. The checker therefore explores all crash/restore orders and sealed-state clones, but does not silently claim a hardware anti-cloning property for live RAM.

Combining, $M \le |\mathcal{C}| \le A$ and $R \succeq \sum_o \rho(o)$ at every reachable state. □

Admission is on the cost, not the count. The same asymmetry has a liveness counterpart. The reserved-count ceiling $n_{\max}$ is calibrated for units priced $\rho(q, \sigma)$; once a step may cost $\rho(1, \sigma)$, "n below $n_{\max}$" is no longer the same policy as "ε below the budget". DP-ATTEST therefore admits a surcharged step against the complete projected vector. Under the strict contract a release at index i owes $m_i \in \{0, 1, 2\}$ surcharge units — one for the repeat, and one more the *first* time its sample context is repeated, to upgrade that context's original subsampled charge — so with prepaid capacity left the projection is $R + m_i\Delta$, and when a new window is needed it is $R + k'\rho(q, \sigma) + m_i\Delta$, with k' shortened until the vector converts below the run's budget at its declared δ. It refuses if even $k' = 1$ does not fit — before any capability is minted, so a refusal costs nothing and releases nothing. Two null cases are covered by targeted regressions: checking the surcharge alone and adding a window afterwards still breaches near the boundary, and admitting a first repeat at one Δ rather than two admits a release whose completed charge then exceeds the budget.

That refusal is what a privacy filter would need *operationally*, and it is the reason the gate is not a mere running total; it is not, on its own, the distributional hypothesis a filter takes as input, which Section B.1 explains we do not have. Without this cost-aware admission, a retry could pass a count test and breach the budget it was authorised under.

What the proposition buys, and what it costs. Repetition becomes post-processing rather than a second release. A capability presented t times is refused whenever the plane still holds its redemption and otherwise yields the bytes it yielded before, so across all t presentations the adversary's view is a function of one release and composition sees one. Note which of the two branches carries the argument: refusal is the easy case and the one an implementation gets for free, while the other is the case a rollback produces and is exactly where the replay defect lived. That is the same distinction ELEPHANTDP draws by storing the answer (Jin et al., 2024), moved from a query service to SGD rate. The price is that condition (iv) makes the mechanism's randomness pseudorandom under a sealed seed, so the guarantee is computational rather than information-theoretic. That is not a new assumption here — the sampler's amplification already rests on β staying secret from the party the guarantee is claimed against — but it is an assumption, and a deployment that wants information-theoretic noise must buy repetition-safety the other way, with a durable one-time redemption record or by forfeiting the whole remaining window on every recovery.

What it does not cover. Three things, and none is hidden elsewhere. *Faithfulness*: (q_i, σ_i, C_i) are what the shim declared, and DP-ATTEST does not bind the noise path (Section 3); the proposition bounds how many outputs were released, not that each was the mechanism it claimed. *Retention* on the record plane: records already handed over cannot be un-handed, so there the certificate binds fresh access and the proposition's M counts batches, not invocations. *The accountant's own measurement*: compromise it and there is no argument left, which is why it is small.

From the count to a privacy statement (ε, δ). Proposition 1 counts distinct outputs and prices them; it is not itself a DP theorem, and we state the gap rather than paper over it. A privacy statement for the primary plane follows from the proposition, an explicit *ideal experiment*, three named assumptions, and a restriction on the schedule. *(F, faithfulness)* each released output is the

declared mechanism at its recorded (q_i, σ_i, C_i) — the noise path is not bound by DP-ATTEST and is what VERIDP supplies in zero knowledge (Abdolmaleki et al., 2026). *(S, sampling)* each context's batch is a Poisson(q) sample over the run's slots, drawn independently of every other context and of every public value — true of the ideal experiment below *by construction*, and of the deployed transcript only through the sampler hybrid H_S, whose assumption ($S^\star$) is stated there. *(P, the public convention)* the experiment must say what is public before it can call anything data-independent; the one item that needs care, the dataset commitment, is a function of the data, and Section B.1 makes it hiding. The anchor generations and frontier count transitions and units, so they are fixed by the schedule and independent of the dataset.

The fixed-schedule statement, in one place. We give the global statement, rather than a sum of per-group marginals, together with the experiment it is a statement about and its quantifiers.

Schedule and release order. Fixed before the run: the sequence of releases — for each, its sample context and its mechanism parameters, which registration pins. Let $\mathcal{K}$ be the contexts used and $m_\kappa \geq 1$ the number of distinct outputs released on κ. Queries — model states — may depend on earlier outputs; a capability's query is fixed when it is authorised, which precedes its release. The **release order** is the order in which distinct outputs cross the plane's boundary; a capability presented again returns the bytes it returned before and is not a release. The schedule is **non-interleaved** if, in the release order, each context's releases are consecutive. It is the release order that matters and not the order of authorisations: a held capability is released later than it was charged, and only releases enter the adversary's view. Per-context counts do not determine it — $(\kappa, \kappa, \kappa')$ and $(\kappa, \kappa', \kappa)$ have the same counts.

Sample identity. A *context* names one hidden sample; a **capability** names one mechanism output, because the capability seeds the noise. They coincide for one replica and do *not* when replicas share a context: each redeems its own capability and receives its own noised output over the same records, so m_κ counts *outputs on* κ, not replicas (Section A.4 explains why the pair is not one application). Independent replicas carry the replica identity in κ.

The ideal experiment. $\mathrm{view}^{\mathrm{ideal}}(D)$ is generated as follows, for the fixed public capacity N of Section B.1.

(I1) For each context $\kappa \in \mathcal{K}$, at its first release, draw an inclusion vector $J_\kappa \in \{0,1\}^N$ with i.i.d. Bernoulli(q) coordinates, independently of everything else; the batch B_κ is the set of filled slots it selects.

(I2) For each distinct capability t on context $\kappa(t)$ at query u_t, release $Y_t = \sum_{x \in B_{\kappa(t)}} \mathrm{clip}(g_x(u_t)) + Z_t$ with $Z_t \sim \mathcal{N}(0, \sigma^2 C^2 I)$ fresh and independent (the plane divides it by the public expected size qN, which is post-processing); presenting t again returns the same Y_t.

(I3) Every public value the deployed transcript derives from the sampler's secret — the seed commitment $H(\beta)$ and the batch commitment each record and capability carries — is an independent uniform string, one per context for the batch commitment (so the equality of commitments across repeats of one context, which the schedule already makes public, is kept); $H(\nu)$ is the hash of a nonce independent of everything else, as deployed; and the dataset root is computed from uniform leaves.

(I4) Anchor generations, frontier, certificate and the capability run/replica/index fields are the deterministic functions of the schedule they are in the deployed run.

So in $\mathrm{view}^{\mathrm{ideal}}$ every published value other than the outputs is independent of D *and* of every J_κ.

Corollary 1 (Fixed, non-interleaved schedules)**.** *Assume (F), (P), and a schedule fixed before the run and non-interleaved in its release order. For neighbouring $D' = D \cup \{x\}$ and every order α,*

$$D_\alpha\big(\mathrm{view}^{\mathrm{ideal}}(D') \,\big\|\, \mathrm{view}^{\mathrm{ideal}}(D)\big) \;\leq\; \sum_{\kappa \in \mathcal{K}} c_\kappa, \qquad c_\kappa = \begin{cases} \rho_\alpha(q, \sigma), & m_\kappa = 1, \\ m_\kappa\, \rho_\alpha(1, \sigma), & m_\kappa > 1, \end{cases}$$

which is exactly what the strict contract charges. Converting at the declared δ gives the certificate's (ε, δ) for the ideal view.

Proof. By non-interleaving the releases form consecutive blocks $G_1, \ldots, G_L$ in release order, one per context. Write $\mathcal{T}_{<\ell}$ for the view before block ℓ, public values (I3)–(I4) included, and κ_ℓ for the block's context.

The block's sample keeps its prior law under conditioning. In the ideal experiment $\mathcal{T}_{<\ell}$ is a function of D, of the inclusion vectors and noise of *earlier* contexts, and of values independent of every inclusion vector (I3–I4). J_{κ_ℓ} is drawn independently of all of them (I1), and because the schedule is non-interleaved no earlier release reads κ_ℓ. Hence $J_{\kappa_\ell} \perp \mathcal{T}_{<\ell}$, and its law given $\mathcal{T}_{<\ell} = \tau$ is the prior for every τ. This is a property of the ideal experiment *by construction*; it is not derived from the sampler's key-derivation function, whose outputs share one secret β and are only computationally independent (the sampler hybrid below). Note what is *not* claimed: the *releases* of block ℓ are not independent of $\mathcal{T}_{<\ell}$, because their queries may be chosen from it.

Each block is a conditional group on a fresh sample. Fix τ. The block's queries are functions of τ and of the block's own earlier outputs, so conditionally the block is an m-release group on a fresh Poisson sample with adaptive queries inside it. For $m > 1$, quasi-convexity over x's single inclusion indicator bounds it by its $I{=}1$ branch, m adaptive Gaussians of sensitivity one, which compose to $m\rho_\alpha(1, \sigma)$; for $m = 1$ the query is fixed by τ and the release is the subsampled Gaussian, $\rho_\alpha(q, \sigma)$. The bound is uniform in τ, so adaptive Rényi composition over the blocks sums them (Mironov, 2017). The values of (I3)–(I4) are independent of D and of every inclusion vector and add nothing. □

A tempting shortcut, and the example that rules it out. It is tempting to argue that "index j's batch is independent of every release at every other index, so conditioning on the rest of the transcript leaves group j with its prior law." The conclusion is true for the past; the stated reason is not, and a two-step example shows why. On a fixed dataset holding one record a, draw $J_1, J_2 \sim \mathrm{Bern}(1/2)$ for a's inclusion at two contexts and $Z_1, Z_2 \sim \mathcal{N}(0, 1)$; fix the schedule $(1, 2)$; let the first clipped contribution be 1 and the second $\mathbf{1}\{Y_1 > 0\}$. Then $Y_1 = J_1 + Z_1$ and $Y_2 = J_2\mathbf{1}\{Y_1 > 0\} + Z_2$, and

$$\mathbb{E}[Y_2 \mid J_1{=}0] = 0.25, \qquad \mathbb{E}[Y_2 \mid J_1{=}1] = \tfrac{1}{2}\Phi(1) \approx 0.420672,$$

so $\mathrm{Cov}(J_1, Y_2) \approx 0.042668$: the first batch's inclusion bit and the second *release* are dependent. **Fixing the schedule is not fixing the queries.** The proof above conditions on the past and uses only $J_{\kappa_\ell} \perp \mathcal{T}_{<\ell}$, which the example does not touch. `test_the_group_independence_sentence_is_refuted` pins it.

Why non-interleaving is a hypothesis, and which runs meet it. With output-dependent queries and an interleaved order such as $(\kappa, \kappa', \kappa)$, grouping the κ-releases is not a chronological composition: the second is chosen after seeing κ''s output, so the group cannot be treated as one conditional mechanism at one point of the sequence. We do not claim that case. The plausible route is a concurrent or interactive composition theorem — Lyu (2022) proves one for interactive mechanisms — applied to a uniform interactive guarantee per context; we have not established that our construction meets its conditions. Two cases are covered without it: a schedule whose queries are fixed in advance (the transcript then factorises over contexts and Rényi divergence is additive over products, so order is irrelevant), and any run whose repeats of a context are consecutive — the step that failed, retried at once. **Restoring a checkpoint and replaying a run of steps is not such a run**: it re-enters contexts after others and is interleaved. DP-ATTEST still prices it — the bookkeeping of Proposition 1 holds for every interleaving — but the corollary does not certify that price as a bound on the adaptive release. Which reported results meet the hypothesis is in the result-mode inventory (Section D.2): the training results contain no repeated context and meet it trivially; the replay row of Table 4, the checker's re-execution campaigns and several traces of Table 9 do not, and are bookkeeping evidence.

Enforcing the schedule: the fresh-once registration. Registered fresh-once (`repeat_policy="refuse"`), the accountant refuses every authorisation on a sample context it has already authorised (Algorithm 1, line 13) instead of pricing it. Let $T^\star$ be the largest count the gate can admit without a surcharge: the largest $t \le n_{\max}$ with $\varepsilon(t\,\rho(q, \sigma)) \le \varepsilon_{\max}$ at the gate's order or orders. It is a function of registered parameters alone, and the certificate carries it with (q, σ) and the replica mode.

Corollary 2 (Fresh-once runs lie inside Corollary 1)**.** *Assume (F), (P) and conditions (i)–(vii) of Proposition 1, with the accountant registered fresh-once under a budget $\varepsilon_{\max}$. Then in every interleaving of authorisations, releases, crashes, restores and repeated redemptions, (a) no sample*

context carries more than one distinct mechanism output, so the release order is non-interleaved; and (b) for every order α, $D_\alpha\big(\mathrm{view}^{\mathrm{ideal}}(D') \,\|\, \mathrm{view}^{\mathrm{ideal}}(D)\big) \le T^\star \rho_\alpha(q,\sigma)$, so the run satisfies $(\varepsilon(T^\star),\delta)$-DP in the ideal experiment with $\varepsilon(T^\star) \le \varepsilon_{\max}$, and computationally so for the deployed view under the hybrids below.

Proof. (a) A capability exists only if the accountant minted it, and under fresh-once it mints one only after the repeat test finds the context unused. That test reads the high-water marks D, which recovery raises through every index a lost window could have used, since the register's frontier bounds them; so no crash, restore or rollback returns a used context to the unused side. This is the fact the proposition's pricing already rests on, and the checker's fresh-once campaign tests it directly (Section 4.3). A capability presented again returns the bytes it returned before (condition (iv)). Each context therefore carries at most one distinct output, and a release sequence in which no context occurs twice is non-interleaved.

(b) Let $S^\star$ be the schedule that releases $T^\star$ fresh contexts, one output each, with adaptively chosen queries. It is fixed before the run and non-interleaved, so Corollary 1 bounds it by $T^\star\rho_\alpha(q,\sigma)$. Every fresh-once run is a post-processing of $S^\star$. A simulator interacting with $S^\star$ assigns the run's t-th minted capability to $S^\star$'s t-th slot, which is sound because in the ideal experiment a fresh context's sample is drawn at its first release independently of its label (I1). It submits the query that capability names, holds the output until the run redeems it, and discards the outputs of slots the run never redeems: forfeited units, capabilities never presented, and every slot after the run stops. The run mints at most $T^\star$ capabilities, because each spends one unit and the gate admits at most $T^\star$ units when no surcharge arises. A query is fixed when its capability is minted, and so depends only on outputs of earlier slots, which the simulator already holds. The anchor values, frontier, certificate and capability fields are functions of the run's own actions and not of the data (I4), so the simulator computes them too. The run's view is thus a function of $S^\star$'s view and of the simulator's coins, and Rényi divergence does not increase under post-processing. □

From the ideal experiment to the deployed transcript: three hybrids. Corollary 1 is an information-theoretic statement about $\mathrm{view}^{\mathrm{ideal}}$. The deployed view $\mathrm{view}^{\mathrm{real}}$ differs in three places, and each hop names what it assumes.

- H_S **Sampler and seed commitments.** Replace every value derived from the sampler secret β — the sampler stream the plane expands into batches, the batch commitments c_κ carried in records and capabilities, and the published $H(\beta)$, the families of the table below — by independent uniform strings, keeping the beacon and $H(\nu)$. Uniform streams give (I1), with the geometric-gap expansion of `Beacon.indices` idealised as exact; uniform commitments give (I3). The hop is assumption **(S*)**, stated with its oracle inputs and their domain separation in the next paragraph.
- H_L **Leaves.** Replace the keyed-HMAC leaves by uniform strings; a distinguisher is a PRF distinguisher at N queries (Section B.1).
- H_N **Noise.** Replace the capability-derived noise stream by true Gaussian noise; a distinguisher is one for the generator under a sealed seed, assumption (iv).

H_N is $\mathrm{view}^{\mathrm{ideal}}$. For every efficient $\mathcal{A}$ and every D,

$$\big|\Pr[\mathcal{A}(\mathrm{view}^{\mathrm{real}}(D))=1] - \Pr[\mathcal{A}(\mathrm{view}^{\mathrm{ideal}}(D))=1]\big| \;\le\; \mathrm{Adv}^{S^\star} + \mathrm{Adv}^{\mathrm{prf}} + \mathrm{Adv}^{\mathrm{prg}} \;=:\; \nu_\lambda,$$

so the corollary's (ε,δ) transfers as

$$\Pr[\mathcal{A}(\mathrm{view}^{\mathrm{real}}(D'))=1] \;\le\; e^{\varepsilon}\Pr[\mathcal{A}(\mathrm{view}^{\mathrm{real}}(D))=1] + \delta + (1+e^{\varepsilon})\,\nu_\lambda$$

— computational differential privacy (Mironov et al., 2009) with $\delta + \mathrm{negl}$. The Rényi inequality itself does not transfer.

Why the sampler hop is not optional. A value whose marginal law does not depend on the data can still destroy amplification jointly. For neighbouring $\{\}$ and $\{x\}$ and one scalar release — $Y = Z$ under the first and $Y = J + Z$ under the second, with $Z \sim \mathcal{N}(0,\sigma^2)$ and $J \sim \mathrm{Bern}(q)$ — a hidden J gives $D_2 = \log(1 + q^2(e^{1/\sigma^2} - 1)) = 1.7\times10^{-6}$ at $q = 10^{-3}$, $\sigma = 1$; published beside Y, the same data-independent J gives $\log(1 + q(e^{1/\sigma^2} - 1)) = 1.7\times10^{-3}$, about 999× larger (`test_p`

ublishing_the_inclusion_bit_forfeits_amplification reproduces both values by quadrature as well as in closed form). This is not a claim that $H(\beta)$ reveals any inclusion bit; it is why every public value correlated with the sampler has to sit inside a hybrid rather than be assumed data-independent. PCG64, which the prototype uses for noise, does not instantiate (iv), so for the artifact as run H_N is assumed, not reduced; what the prototype demonstrates is the bookkeeping of Proposition 1 (Table 2).

Assumption (S*), with its oracle inputs. Every value derived from the sampler's secrets is an evaluation of one hash H at an encoding $\langle x_1, \ldots, x_m \rangle$ that prefixes each part with its 32-bit length. The encoding is injective on tuples of every arity, so two evaluations share an input only if they agree in arity and in every part. There are five families (Table 14; tags carry the prefix dp-attest/ in the artifact). Here b is the beacon, j' the replica in independent mode and empty

Table 14: The five oracle families of (S*), and their oracle inputs.

Family	Oracle input	Published
seed β	$\langle$beacon-seed, b, $\nu\rangle$	no
nonce commitment $H(\nu)$	$\langle$nonce-commit, $\nu\rangle$	at registration
seed commitment $H(\beta)$	$\langle$beacon-commit, $\beta\rangle$	yes
sampler stream $\mathrm{KDF}(\beta, \kappa)$	$\langle \beta, rid, i, j', c\rangle$	no; expanded into B_κ
batch commitment c_κ	$\langle$batch, β, rid, i, j', $B_\kappa\rangle$	in records and capabilities

in shared mode — so (rid, i, j') is the sample context $\kappa(j, i)$ — and c a block counter. The two two-part families differ in their tag and every other pair in arity, so the families are disjoint (test_the_sampler_oracle_families_are_disjoint_inputs). **(S*)**: *for a uniform λ-bit ν with $\lambda \leq 256$, any public beacon b and any adversary making Q oracle queries, $H(\beta)$ together with the stream and batch-commitment functions of β is indistinguishable, given b and $H(\nu)$, from a uniform string and two independent random functions, with advantage at most $2Q\,2^{-\lambda}$.* In the random-oracle model this holds. Until the adversary queries H at an input containing β, every value above is an independent uniform string, because the families' inputs are disjoint from each other and from any query made without β. Such a query needs β, which the adversary obtains only by querying $\langle$beacon-seed$, b, \nu\rangle$ at the right ν — each query tests one candidate, whether against $H(\nu)$ or through the seed family, so at most $Q\,2^{-\lambda}$ — or by hitting β directly, at most $Q\,2^{-256} \leq Q\,2^{-\lambda}$. We do not reduce (S*) to a standard-model assumption. Ordinary PRF security is not enough, because the construction publishes a hash of the very key the batches are derived from.

What the verifier checks, and what it cannot. It checks the pinned parameters, the per-mechanism counts, the chain heads, the anchor value and the recomputed ε: the bookkeeping of Proposition 1. Whether it can also check Corollary 1's premises depends on the registration, which the certificate names (repeat_policy). Registered **fresh-once**, the premises are enforced rather than claimed. The certificate carries the committed schedule ($T^\star$, q, σ, replica mode); the verifier recomputes $\varepsilon(T^\star)$ from it, and rejects a fresh-once certificate that shows a surcharge unit or a duplicate (verify_certificate). Corollary 2 then gives the privacy statement. Registered with **priced repeats**, the verifier cannot check either premise: counts cannot tell $(\kappa, \kappa, \kappa')$ from $(\kappa, \kappa', \kappa)$, and an adaptive schedule produces a certificate indistinguishable from a predeclared one. Both premises are then *deployment obligations*, and the certificate claims bookkeeping integrity and a recomputable charge only. The plane records its ordered release log (DataPlane.release_log), and our checker reads it to report which traces meet the hypothesis (Table 9, last column). We did not put that log into the certificate. Under fresh-once it is unnecessary; under priced repeats it would have to be anchored at every release, since the plane's own state is as rewindable as the accountant's, and a rolled-back log could omit the very release that made an order interleaved.

Why the commitment cannot simply be a published Merkle root. A natural design publishes a plain binding root and then holds it *fixed* across the neighbouring pair. That is not a convention; it is a different experiment, and the statement it stands in for is vacuous. Take $D' = D \cup \{x\}$ under the add/remove adjacency of Section 2, let $r(\cdot)$ be the deterministic root, and consider the event $E = \{\text{published root} = r(D)\}$. Then $\Pr[\mathcal{M}(D) \in E] = 1$ and $\Pr[\mathcal{M}(D') \in E] = 0$, so the DP inequality forces $1 \leq \delta$ at every finite ε. No hash inversion is involved: it follows from quantifying over all events and all neighbouring pairs, and collision-resistance is the wrong property to appeal to

— binding is exactly what makes the root a perfect distinguisher. Fixing the root instead excludes precisely the pairs the adjacency ranges over, so it cannot stand in for the statement it was substituted into.

What we do instead, with the assumption named. The default commitment is *hiding*, and hiding is a property of a keyed primitive rather than of a hash, so the construction is keyed. Fix a *public capacity* N for the run. A leaf is

$$\ell_i \;=\; \begin{cases} \mathrm{HMAC}_s(\texttt{0x00} \,\|\, \langle i \rangle \,\|\, x_i) & i < |D|, \\ \mathrm{HMAC}_s(\texttt{0x02} \,\|\, \langle i \rangle) & |D| \le i < N, \end{cases} \qquad i = 0, \ldots, N-1,$$

under a uniform 256-bit per-run salt s generated inside the accountant's confidential VM, never released, and sealed to the plane's measurement exactly as the noise seed in (iv) is. **The assumption is that** HMAC**-**$\mathrm{SHA256}$ **with a secret uniform key is a pseudorandom function.** The reduction is then the usual one: an adversary distinguishing the published root for D from the one for D' gives a distinguisher for HMAC_s from a random function at N queries, since the root is a fixed function of $(\ell_0, \ldots, \ell_{N-1})$ and the N PRF inputs are distinct by construction. Collision resistance is *not* the property being used: it says a root binds, and says nothing about what it hides.

Two details the reduction needs. The leaf is bound to its *position* $\langle i \rangle$: a PRF's outputs are distinct only on distinct inputs, so two identical canonicalised records would otherwise produce one repeated leaf — and a repeated leaf is a pattern no uniform vector has, which is exactly what the reduction claims the leaves look like. And the tree has a *fixed public capacity*: add/remove adjacency changes $|D|$, so a tree whose leaf count and shape follow $|D|$ leaks the bit the adjacency ranges over however well each leaf hides. With both, the published root is a function of s and a length-N vector, N is a declared parameter of the run rather than a property of the data, and the neighbouring pair D, $D \cup \{x\}$ is admissible with both roots published. The sampler draws over the N slots at rate q and a slot no record fills contributes nothing, which is the "Poisson subsampling over a fixed universe" the accounting already models — so the capacity costs amplification only to the extent that N exceeds $|D|$, and a deployer choosing N is choosing how much of the corpus size to hide. A deployment that cannot fix N in advance is in the same position as one publishing $|D|$, and should say so rather than rely on this reduction.

The adversary-visible transcript, in full. What the privacy adversary sees is: the M released mechanism outputs; the anchor's generation values and its accounting frontier (B, n, s); the published $H(\beta)$ and $H(\nu)$; the hiding root; the certificate, which names the accountant's measurement, the per-mechanism accounted counts and the reported (ε, δ); and the run, replica and index fields of every capability, with the batch commitment and query digest it carries — the commitment is keyed by β, which is why the sampler hybrid H_S replaces it. It does *not* include s, ν, β, the plane's noise seed, the MAC key, or any sealed blob. The generation values and the frontier count transitions and units, not records, so they are data-independent outright; the root is computationally independent by the assumption above; the rest is the transcript Proposition 1 bounds.

Who may hold the salt. The plane needs s to build and check the tree; the accountant generates it. An auditor given s can open any leaf and is therefore *inside* the trust boundary — not a member of the adversary coalition the corollary quantifies over. Releasing s to a party in that coalition collapses the commitment to the transparent case, and the certificate should therefore name the salt's holders, because a deployment that hands s to the same party that receives the model has published a binding root by another route.

The verifier loses nothing it had: the root still binds the corpus the run was registered against, the plane still checks a batch commitment against it, and the auditor above can still open any leaf. What changes is the strength of the conclusion, which becomes computational: the adversary's view is *indistinguishable* from one containing no root rather than identical to it, so the guarantee is computational (ε, δ)-DP in the sense of Mironov et al. (2009). DP-ATTEST already needs that regime for condition (iv), where the mechanism's noise is a PRF under a sealed seed, so this adds a reduction rather than an assumption of a new kind — and the prototype's PCG64 stream is not a cryptographic PRF, so the implementation exercises the bookkeeping and not the computational-privacy conclusion (Section C.2).

Two alternatives, named rather than blurred. A deployment that must keep a plain binding root may withhold it from the adversary's view — releasing it only to an auditor, which makes it part

of a trusted verification channel and not part of the transcript the composition is over — or may publish it and accept a *different* adjacency, in which neighbours share the committed corpus and differ in which records the run may *use*. That statement is well-formed and may be the right one for a registered-dataset regime, but it is not example-level add/remove, and we do not report it as though it were. The distinction is narrow: under the shared-root adjacency nothing is claimed about a record's presence in the corpus, only about its contribution to the run.

Under (F), (S) and (P), the adversary's whole view is the M released outputs plus state that is constant, data-independent, or (for the hiding root) computationally independent of the data in the experiment, and by Proposition 1 $M \le A$ with R dominating their summed costs; composing those per-release bounds over the accounted trace, and treating iterates, checkpoints and the released model as post-processing of that transcript (sole custody, ii), gives an (ε, δ) for the released model no worse than the accountant's reported value. The corollary is deliberately conditional: DP-ATTEST enforces the count, the cost domination and the completeness/monotonicity that make A and R meaningful; (F) is the mechanism-fidelity assumption a full certificate still needs and which Section 4 does not claim to have discharged, (S) is discharged by the unamplified re-execution charge below, and (P) is a statement about the experiment rather than about the code. A reader who grants none of the three still gets the accounting-integrity statement of Proposition 1, which is what the evaluation measures.

The retry and sampling contract (S), made explicit. Assumption (S) matters only because a re-executed index reuses a hidden sample, and reusing one inclusion draw across several noised releases is not the same joint mechanism as fresh subsampling each time — for a one-record neighbouring pair with clipped scalar contribution 1, inclusion $I \sim \mathrm{Bernoulli}(q)$ drawn once and reused for T releases with independent noise gives Rényi-2 divergence $\log\big(1 + q^2(e^{T/\sigma^2} - 1)\big)$, above the $T\log\big(1 + q^2(e^{1/\sigma^2} - 1)\big)$ of T independent draws (at $q = 10^{-3}, \sigma = 1, T = 2$: 6.39×10^{-6} versus 3.44×10^{-6}). Charging a re-execution as a fresh subsample would therefore under-account. DP-ATTEST resolves this at the authorisation interface, and the two cases are distinct by construction. A capability *presented* again is exact retransmission: the plane returns the object it already returned (Proposition 1), no new release, nothing to account. A capability minted on a sample context that has already been authorised — a re-execution, or a second replica sharing the context; the only ways a new output reaches the same sample — is charged *unamplified*, at $\rho(1, \sigma)$ rather than $\rho(q, \sigma)$ (Algorithm 1, line 12).

What discharges (S) is the cumulative charge, not a per-release one. A single $\rho(1, \sigma)$ does not upper-bound the reused-inclusion divergence for every T: at $q = 10^{-3}$, $\sigma = 1$, $T = 20$ the divergence above is 6.1865 against a single unamplified order-two charge of 1. The correct statement is about the *total* charged across a reused sample: by quasi-convexity of Rényi divergence (van Erven & Harremoës, 2014), the mixture over the inclusion indicator is dominated by its $I = 1$ branch, which is T independent Gaussians on one record, so

$$D_\alpha\big(T \text{ releases on one reused inclusion}\big) \;\le\; T\,\rho_\alpha(1, \sigma) \qquad \text{for every } \alpha,\, q,\, \sigma,\, T.$$

DP-ATTEST charges exactly that: a release on a context the run has already authorised pays $\rho(1, \sigma)$, and the *first* repeat on a context additionally upgrades that context's original subsampled charge, so the cumulative total after T releases is $T\rho(1, \sigma)$ and (S) needs no condition on (q, σ, T). The upgrade costs one Δ per context ever repeated, which is why it is recorded per context — not per release, and not per replica. A deployment may instead select the `amplified-first` contract, which keeps the first release's subsampled price and so charges $T\rho(1, \sigma) - \Delta$; that is sound at order two for every (q, σ, T) — the charge and the divergence coincide at $T = 1$ and the charge's slope $1/\sigma^2$ strictly exceeds the divergence's $(x/\sigma^2)/(1 - q^2 + x)$ with $x = q^2 e^{T/\sigma^2}$, so the gap only opens — but the closed form above is written at order two alone, and registration verifies the inequality numerically over the budget's range before admitting that contract. The strict contract is the default because the quasi-convexity bound holds at every order in the grid with nothing to check.

A per-round conditional bound, and a counterexample to it. A natural route to adaptive revisiting is a *per-round conditional* bound: that for every transcript $y_{<t}$, the next release costs at most $\rho_\alpha(q, \sigma)$ at a fresh index and $\rho_\alpha(1, \sigma)$ at a repeated one. It would license the Rényi filter over adaptive revisiting and `amplified-first` pricing at every order. **The bound is false**, and we give the counterexample because the failure is instructive.

Take $D = \{a\}$, $D' = \{a, x\}$, $q = 0.1$, $\sigma = 0.2$, clipping bound one. Draw $J, I \sim \mathrm{Bern}(q)$ once, for a's and x's inclusion in the single batch. Fix two queries in advance: the first scores $+1$ for both records, the second $+1$ for a and -1 for x. Both respect the clipping bound, and each release draws its own noise, so

$$D:\ (Y_1, Y_2) = (J + Z_1,\ J + Z_2), \qquad D':\ (Y_1, Y_2) = (J{+}I{+}Z_1,\ J{-}I{+}Z_2).$$

Condition on $Y_1 = 1$. Under D this forces a in: the posterior on $J{=}1$ is 0.999966. Under D' the same observation is equally well explained by $(J, I) = (1, 0)$ and by $(0, 1)$, each with posterior 0.499992. The conditional law of Y_2 is therefore concentrated near $+1$ under D but is close to an even mixture of $+1$ and -1 under D'. Taking $E = \{Y_2 \le -1.6\}$,

$$P(E) = 2.086433 \times 10^{-20}, \qquad Q(E) = 6.749376 \times 10^{-4},$$

so by Cauchy–Schwarz $D_2(Q\|P) \ge \log\big(Q(E)^2/P(E)\big) = 30.7145$, against the claimed $\rho_2(1, \sigma) = 25$.

Where the proof broke. It asserted that under D the conditional law of the next release is a single distribution P "regardless of the inclusion variables", and then wrote the D' side as $(1 - \pi)P + \pi Q$ with *that same* P. Conditioning on the transcript changes the posterior over the *other* records' inclusion, and it changes it *differently in the two worlds*: here $Y_1{=}1$ pins $J{=}1$ under D but not under D'. The two branches then differ by the *sum* of the two records' contributions — an effective shift of 2, not the sensitivity 1 — and no per-release unamplified bound can hold. Quasi-convexity was applied to a mixture pair that does not share a common component.

What survives, and why. The *group* argument does, and it is the one the strict contract has always rested on. Before any conditioning, the law of the other records is the *same* on both sides, so the transcript of all T releases at one index is a mixture over the single inclusion indicator with a genuinely common component, and quasi-convexity applies to it. In the counterexample the full two-release transcript has $D_2(D'\|D) = 45.39$ and $D_2(D\|D') = 0.11$, both under the strict charge $2\rho_2(1, \sigma) = 50$ — the price DP-ATTEST actually pays. Section B.1 records both computations, and `test_the_per_round_conditional_bound_is_refuted` pins the counterexample.

What does not follow. `amplified-first` has no all-orders justification; it rests on the order-two closed form above, with registration verifying it numerically over the budget's range, and that check is load-bearing. And the corollary's adaptive case is **not claimed**; the scope paragraph below says what is and is not supported.

Scope of the privacy corollary: fixed, non-interleaved schedules. The quasi-convexity argument above is stated for a *fixed* group of T releases on one sample context: it compares two mixtures over one inclusion indicator, and T is a parameter of that comparison. Within that group the queries may be chosen adaptively, since for a fixed inclusion the releases compose adaptively at $\rho_\alpha(1, \sigma)$ each (Mironov, 2017) — and the counterexample above fixes its two queries in advance, so it is not a counterexample to this. What the group argument does not give is an adaptively chosen *number* of releases.

Summing realised per-release charges over a sequence whose length depends on the outputs is the composition Rogers et al. (2016) show is invalid in general; it needs a privacy *filter*, and Feldman & Zrnic (2021) give the Rényi form. A filter takes as its hypothesis a bound on the *next* release conditional on the reports so far, which is exactly the per-round bound refuted above. **We do not have that hypothesis, and we therefore do not claim the adaptive case.** The privacy corollary is scoped to runs whose schedule — the number of releases, their contexts and their parameters — is fixed before the run *and* non-interleaved in release order (Corollary 1). Every reported schedule is fixed, crash points included; which of them are also non-interleaved, and therefore inside the corollary rather than only inside Proposition 1, is listed in the result-mode inventory (Section D.2). No reported number depends on the per-round bound. The fresh-once registration is not the adaptive case in this sense, and does not need the missing hypothesis: its certified value is the *committed* schedule's $\varepsilon(T^\star)$, never a sum of realised charges, and a run that skips or stops early is a post-processing of that schedule (Corollary 2(b)). What stays unclaimed is certifying the realised, smaller cost of an adaptively stopped run.

Two parts of the machinery do not depend on the per-round bound, because both are about *enforcement* rather than about a distribution.

First, the gate does halt before the budget, counting the complete price: condition (v) puts base units, surcharge and the strict first-repeat upgrade inside the admission test at line 15 rather than after it, so no release is minted against a budget its finished charge then exceeds. That is a property of the implementation, tested by `test_the_first_repeat_upgrade_enters_the_admission_check`, and it is what a filter would need *operationally*. It is not what a filter needs *distributionally*.

Second, pinning the order is right and necessary. A filter is fixed at one order against one budget, while an accountant ordinarily converts at whichever order of the grid is smallest; under a data-dependent schedule that chooses the order with the data in hand. A run may therefore declare its order at registration (`filter_order`): the gate tests that coordinate alone, `epsilon` reports at it, and the certificate carries it so the verifier checks the statement the run enforced. Pinning is not free — every order carries a floor of $\log \frac{\alpha-1}{\alpha} - \frac{\log \delta + \log \alpha}{\alpha-1}$, so a badly chosen order can refuse the first step. **But pinning is necessary, not sufficient**: it fixes which coordinate is enforced and does nothing to supply the conditional bound that is missing. `test_an_adaptively_stopped_run_stays_inside_its_pinned_budget` demonstrates the mechanism, and demonstrating a mechanism is not proving a theorem.

What would close it. Either a conditional bound that survives the posterior shift the counterexample exhibits — plausibly at a larger constant, or under a restriction on how much of the batch a query may depend on — or a group-level privacy-loss process that charges an index's whole retry group when the index is first reopened, which would convert adaptive revisiting into a fixed-T problem at the cost of prepaying retries that may not happen. We have built neither, and we prefer to leave the corollary scoped rather than assert a bridge we cannot prove.

Where that surcharge is and is not exercised. It is paid only on the re-execution path, and Table 6 does not take it: the training harness advances its step index past the accountant's high-water mark on every restart, so its crash rows contain no repeated index at all (the per-run ledger reports `repeat_index_attempts` $= 0$ in every defended cell), so those rows do not carry the charge, and a rounded ε column could not show the difference either way. What exercises it is Table 19, where a replayed index is charged and the understatement of an amplified accountant is visible, together with two targeted regressions: one asserts a re-executed index is charged the unamplified rate, and one crashes immediately after such a release and checks that the surcharge survives an equal-generation recovery (Section A.4).

The per-round counterexample, computed. Both numbers quoted in Section B.1 are reproduced by `test_the_per_round_conditional_bound_is_refuted`, which tests a mathematical claim rather than the implementation: it asserts that the conditional bound fails and that the group bound holds.

The conditional bound fails. With $D = \{a\}$, $D' = \{a, x\}$, $q = 0.1$, $\sigma = 0.2$, the two fixed queries of the counterexample, and conditioning on $Y_1 = 1$: the posterior on $J{=}1$ is 0.999966 under D, while under D' the states $(J, I) = (1, 0)$ and $(0, 1)$ carry 0.499992 each. On $E = \{Y_2 \le -1.6\}$ the conditional laws give $P(E) = 2.086433 \times 10^{-20}$ and $Q(E) = 6.749376 \times 10^{-4}$, so $D_2(Q\|P) \ge \log(Q(E)^2/P(E)) = 30.7145 > 25 = \rho_2(1, \sigma)$. The bound is a Cauchy–Schwarz lower bound on a finite Gaussian mixture, so it needs no quadrature, and by continuity it persists on a set of previous outputs of positive probability rather than at one point.

The group bound holds. For the same construction, the joint law of both releases gives $D_2(D'\|D) = 45.39$ and $D_2(D\|D') = 0.11$ by direct two-dimensional quadrature, against the strict contract's charge for two releases at one index, $2\rho_2(1, \sigma) = 50$. This is the separation that matters: conditioning on part of the transcript is what breaks, and the unconditioned group is what DP-ATTEST prices.

B.2 ANALYTIC ATTACKS: ROLLBACK, EVASION, FORKING AND REPLAY

Rollback by an ordinary preemption. The weakest version of the problem is not an attack. A preemption restores the trainer from its last checkpoint; if the accountant's state is checkpointed alongside — the natural implementation, and the one Table 5 finds — it is restored with it and the steps between the checkpoint and the crash are forgotten. Those steps really touched the data, and the checkpoint training resumes from is downstream of them. No attestation of the trainer binary is in a position to notice, because the binary that ran was the correct one both times.

Table 15: Checkpoint restore that rewinds the accountant: each restore buys 500 unaccounted steps. $q = 0.001$, $\sigma = 1.0$, $\delta = 10^{-5}$, claimed $\varepsilon = 1.0$.

Restores	Steps run	Reported ε	Recomputed ε	Inflation
0	27,399	1.00	1.00	1.00×
1	27,899	1.00	1.01	1.01×
2	28,399	1.00	1.01	1.01×
4	29,399	1.00	1.02	1.02×
6	30,399	1.00	1.04	1.04×
8	31,399	1.00	1.05	1.05×
10	32,399	1.00	1.06	1.06×

Table 16: Rollback as budget evasion: each cycle exhausts the budget, then rewinds.

Cycles	Steps run	Reported ε	Recomputed ε	Inflation
1	27,399	1.00	1.00	1.00×
2	54,798	1.00	1.32	1.32×
4	109,596	1.00	1.88	1.88×
8	219,192	1.00	2.75	2.75×
16	438,384	1.00	4.05	4.05×
32	876,768	1.00	6.05	6.05×
64	1,753,536	1.00	9.15	9.15×
128	3,507,072	1.00	13.87	13.87×

Table 15 reports what that costs: very little. Each restore buys 500 steps of privacy loss that was incurred and never accounted. One restore moves the accountant's own bound for the executed trace from the claimed 1.0 to 1.01, an inflation of 1.01×; ten restores reach 1.06, or 1.1×. An understatement of this size changes no threat model, and we do not argue that it should. The result is here because it occurs without anyone intending it or benefiting from it, which makes it the honest-implementer form of the problem, and because it establishes the primitive the rest of this subsection builds on through a mechanism every production training stack already invokes routinely and logs as normal operation.

Evading the budget outright. A budget-enforcing accountant is a gate: training proceeds until the accountant reports the budget spent. If its state can be restored to an earlier point, the gate reopens, and it can be reopened as often as the operator cares to.

Table 16 runs that loop. Each cycle executes a full budget's worth of steps and then rewinds the accountant, so the reported ε is pinned at 1.0 indefinitely while the executed step count grows without bound. After 128 cycles the run has executed 3,507,072 steps, which the same accountant given that trace bounds at 13.87, an inflation of 13.9×, and the accountant still reports 1.0. The reported number is not stale. It is unrelated to what happened, and no amount of further training will move it.

Two properties should be stated at their true strength rather than at their most alarming. First, the growth is slow. Privacy loss under composition grows roughly with the square root of the executed step count in this regime, so 128 full budgets buy a factor of 13.9 rather than a factor of 128: the violation is unbounded, but it is not a collapse, and an adversary who wants a large multiple pays in GPU-hours. Second, nothing here defeats a cryptographic mechanism. The attestation report verifies at every row, because it measures the trainer binary and the trainer binary is unmodified: it is the correct DP-SGD implementation, applying the correct clip and the correct noise to the correct data, invoked more times than the accountant remembers. A verifying report standing next to a false ε is this paper's thesis in one table.

Table 17 is that loop performed by production software rather than modelled. Each cycle checkpoints, trains, and calls `load_checkpoint`; the accounted step count returns to its pre-cycle value every time while the executed count grows, so after 8 cycles Opacus 1.6.0 has executed 14,068 steps, accounts 500 of them, and reports 1.653 — the same value it reported after the first cycle. Nothing is patched, and every call is one the library documents as its recovery path.

Table 17: Budget-gate evasion through Opacus's documented preemption API: checkpoint, train to the gate, `load_checkpoint`, repeat. Nothing is patched.

Cycle	Steps executed	Steps accounted	Reported ε
1	2,196	500	1.653
2	3,892	500	1.653
3	5,588	500	1.653
4	7,284	500	1.653
5	8,980	500	1.653
6	10,676	500	1.653
7	12,372	500	1.653
8	14,068	500	1.653

Table 18: A shared accountant forked across data-parallel replicas.

Replicas	Steps run	Reported ε	Recomputed ε	Inflation
1	4,000	0.71	0.71	1.00×
2	6,000	0.71	0.74	1.03×
3	8,000	0.71	0.76	1.07×
4	10,000	0.71	0.79	1.10×
5	12,000	0.71	0.81	1.14×
6	14,000	0.71	0.84	1.17×
7	16,000	0.71	0.86	1.20×
8	18,000	0.71	0.89	1.24×

Table 19: Whether an accountant can tell it was rewound, across a preemption that re-executes 6,000 touches.

Accountant	Acct'd	Rollback seen	Rep. ε	Recomp. ε	Under.
running total	5,000	no	0.73	0.74	1.02×
durable step indices	5,000	yes	0.73	0.74	1.02×

A shared accountant: forking. Rollback requires a restore. Forking requires only that the accountant be consulted by more than one process, each believing it is the only one. Data-parallel training, elastic rescheduling, and sweeps over a shared corpus all have that shape, and a multi-tenant accounting service whose clients each restore the same state is a forking attack nobody has to launch.

Table 18 forks a run at a common prefix and lets each replica proceed independently against the same dataset. The accountant composes one branch; the accounting for the union must compose all of them. At eight replicas that recomputes to 0.89 against a reported value unchanged from the single-replica case, an inflation of 1.2×. That is a modest multiple, for a structural reason: the branches are short relative to the shared prefix, so composition over their union grows slowly. What makes the row worth reporting is the failure mode. There is no rollback event to log, no restore to audit, and no anomalous behaviour in any individual replica — each executes an honest DP-SGD run and receives an honest answer to the question it asked. The error is in the question. An accountant that cannot distinguish the eight-thousandth step of *this* run from the eight-thousandth step of *a* run has no way to notice, which is why Section 3 binds accountant state to a dataset-and-run identity. In a multi-tenant service the fork crosses tenants, and the tenant whose budget is overspent is not the one whose job crashed.

Replay: can the accountant tell it was rewound? The preceding paragraphs show an accountant reporting a number that is too small. This one asks the defensive question: is the rewind observable from inside the accountant, and with what state? Table 19 simulates an ordinary preemption — run, crash, restore, re-execute the lost steps, continue — against two accountants that differ in exactly one respect. The first keeps a running Rényi total and a step counter, both checkpointed and therefore both rewound. The second additionally keeps a set of committed step indices that is durable and survives the restore.

Table 20: What kind of evidence each result is.

Kind	Establishes	Where
library measurement	released code, own APIs	Tables 5 and 25
analytic	exact under our accountant	Section B
model audit	lower bounds on leakage	Table 23, Figure 6
system measurement	the defence as built	Tables 4 and 24
exhaustive check	$A \geq E$ at every interleaving	Section 4.3
design consequence	not measured; marked so	Section C.2

Table 21: Records a retention window of W batches leaves reachable, against a 100,000-record corpus.

W (batches held)	1	2	4	8	16	64
Records reachable	98	193	386	769	1,549	6,119
Of corpus	0.10%	0.19%	0.39%	0.77%	1.55%	6.12%
$1-(1-q)^W$	0.10%	0.20%	0.40%	0.80%	1.59%	6.20%

Table 22: Corpus handed to a compliant run by step t (top), and the cost of capping it (bottom).

Steps served t	100	500	1,000	3,000	5,000	10,000	27,399
Corpus held	9.5%	39.3%	63.1%	95.0%	99.3%	100.0%	100.0%
Coverage cap: steps it allows, as a fraction of the honest run							
Cap	0.25	0.5	0.9	0.99	0.999		
Steps allowed	288	697	2,295	4,605	6,871		
Of honest run	1.1%	2.5%	8.4%	16.8%	25.1%		

The headline of this table is the third column, not the fifth. Both accountants report 0.73 where the data was touched 6,000 times and a rolled-back running total accounts 5,000 of them.

The fifth column is priced amplified. It charges every touch at the declared (q, σ), re-executions included, giving 0.74 and an understatement of 1.02×. That is not DP-ATTEST's contract: the table comes from a closed-form comparator model in `experiments.py` that never runs the accountant, so the retry contract does not apply to it. Under the contract this paper actually argues for, the 1,000 re-executed indices draw the *same* beacon-derived batch, their inclusion is not re-randomised, and amplification is forfeit — so the same trace recomputes to 2,010, an understatement of 2,766× rather than 1.02×.

We report both, and we keep the small number in the table, because the two columns answer different questions and the gap between them *is* this paper's thesis: whether a re-executed index keeps its amplification is worth three orders of magnitude here. What this experiment establishes is the third column — visibility — and for that claim the conservative amplified pricing is the right one to display, since it makes the detection result independent of how the repeats are priced. The gap is negligible only under amplified pricing. What differs between the rows is visibility. The running-total accountant sees nothing, and cannot: from the inside, a rewound counter and a fresh counter are the same object, and no self-consistency check on rewindable state can recover the difference. The accountant with durable indices sees 1,000 step indices arrive a second time, which is direct evidence that its Rényi state was rolled back — not an anomaly score or a statistical heuristic, but a commitment it made and is now being asked to make again.

The result is detection, not rejection, and that distinction shapes the defence. A duplicate index must not be refused: the re-executed step genuinely touched the data a second time and its loss must be charged. The primitive is circular unless the circularity is broken explicitly, because the index set is itself accountant state and an adversary who can roll back the accumulator can roll back the index set with it — which is exactly what the external anchor of Section 3.1 is for, and why accountant continuity reduces to state continuity for trusted execution (Parno et al., 2011; Strackx & Piessens, 2016; Matetic et al., 2017; Angel et al., 2023).

Table 23: One auditor applied to three models from one crash-and-resume run, over 12 seeds.

Iterate	Charged	Acc.	$\varepsilon \geq$	95% CI	p
checkpoint	yes	0.5403	0.087	[0.054,0.120]	4×10^{-36}
discarded iterate	**no**	0.5475	0.116	[0.086,0.145]	2×10^{-49}
published model	yes	0.5482	0.118	[0.087,0.150]	1×10^{-50}

Table 24: The anchor under attack, and 200,000 randomised release-token forgeries.

	Adversary's position	Outcome
A1	store rollback, anchor intact	detected
A2	anchor rewound below sealed state	refused
A3	anchor frozen (increment suppressed)	refused at commit
A3b	frozen anchor, then store rollback	refused at commit
A4	anchor quorum unavailable	fail-closed
A5	equivocating quorums (split brain)	detected (certificate collision)
A6	crash inside the commit window	detected
A7	clone from sealed state	prevented (CAS)
A8	disclosed sampling schedule	enforced (unamplified)
A9	data-plane rollback	bounded to k
A10	register carries no accounting frontier	fail-closed
A11	authentic register read replayed after rollback	refused (MAC-only: fooled)
A12	old CAS success replayed to a clone	refused (MAC-only: forks)
A13	register reply delivered out of order	refused (MAC-only: fooled)
A14	nonce restored with the sealed state	refused (state nonce: fooled)
A15	socket-register read replayed after rollback	refused (unauthenticated: fooled)
A16	plane restart fed a replayed register read	refused (MAC-only: fooled)

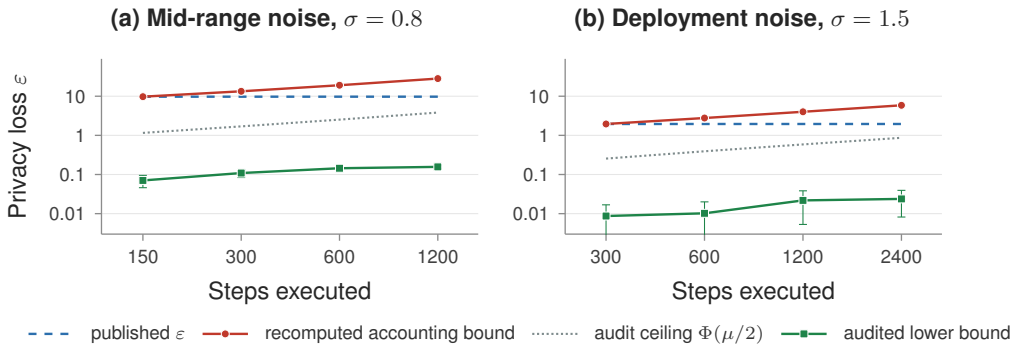


Figure 6: Measured leakage behind an unchanged published ε.

Table 22 is Section 4.2 numerically: what a compliant run has been handed by step t, and what a coverage cap would cost.

Table 23 is Section 4.1 numerically: the same auditor applied to the checkpoint a resume started from, the iterate its crash discarded, and the model finally published.

Table 24 is Section 4.3 numerically, one row per position the anchor was attacked from.

Table 26 is Figure 6 numerically: the audited lower bounds, their 95% intervals, the recomputed and published ε, and the Gaussian-DP ceiling $\Phi(\mu/2)$ that bounds any audit of this form. Within a block the rows differ only in how many cycles the accountant was rewound, so every row publishes the same ε. Section 4.1 interprets them.

Table 27 is Figure 3(b) numerically, over the same sweep of commit interval k and anchor latency, and additionally separates the prefetched from the non-prefetched configuration. Each cell is the **median of 5 passes**; the quoted cell's spread is 14%, measured at load average 13. A single pass is

Table 25: Real DP-SGD under Opacus 1.6.0 that checkpoints, crashes and resumes. Neither crash is adversarial.

	Steps			ε	
Pattern	Touched	Accounted	Lost	Reported	Owed
library-managed	800	600	200	2.647	3.007
caller-managed	800	600	200	2.671	3.007

Table 26: Audited ε in three noise regimes, over 15 seeds. Within a block every row publishes the same ε.

Cycles	Executed	Recomp. ε	μ	Audited $\varepsilon \geq$	Ceiling
Strong signal: $\sigma = 0.4$, reported $\varepsilon = 96.07$ on 300 accounted steps					
1	300	96.07	33.61	0.603 ± 0.058	5.81
8	2,400	402.66	95.06	1.103 ± 0.092	5.81
Mid-range: $\sigma = 0.8$, reported $\varepsilon = 9.73$ on 150 accounted steps					
1	150	9.73	1.52	0.071 ± 0.044	1.16
2	300	13.35	2.15	0.109 ± 0.046	1.70
4	600	19.08	3.04	0.145 ± 0.035	2.52
8	1,200	28.25	4.30	0.157 ± 0.028	3.82
Deployment: $\sigma = 1.5$, reported $\varepsilon = 1.95$ on 300 accounted steps					
1	300	1.95	0.41	0.009 ± 0.015	0.26
8	2,400	5.86	1.17	0.024 ± 0.028	0.87

Table 27: End-to-end overhead as a fraction of a measured step, by anchor latency (columns) and commit interval k (rows).

	Anchor latency (ms)			
k	0	0.2	1	10
On the critical path				
1	3.01%	6.22%	17.15%	122.05%
10	1.72%	2.11%	3.14%	13.64%
50	1.57%	1.63%	1.86%	3.98%
100	1.55%	1.58%	1.69%	2.74%
500	1.54%	1.53%	1.57%	1.80%
Prefetched				
1	1.58%	5.00%	16.59%	127.05%
10	0.35%	0.69%	1.79%	13.17%
50	0.21%	0.27%	0.48%	2.77%
100	0.18%	0.21%	0.33%	1.47%
500	0.16%	0.16%	0.18%	0.43%

Table 28: Calculated overhead (%) of the added per-step cost across measured DP-SGD step times.

				k / anchor latency		
Model	Params	Batch	Step (ms)	100 / 0.2 ms	10 / 1 ms	1 / 10 ms
MLP	0.033M	64	3.3	1.89	15.9	1,124
MLP	0.59M	128	29.4	0.21	1.79	127
MLP	3.3M	256	213.1	0.029	0.25	17.5
transformer	4.2M	32	395.1	0.016	0.13	9.44
transformer	4.2M	128	1,160.3	0.0054	0.045	3.21
MLP	13M	256	1,501.0	0.0042	0.035	2.49

not reproducible to two decimals: at fixed conditions the spread is tens of percent, and the median moves with host load. Every socket call here is authenticated and request-bound (Section A.2).

Table 29: Data-plane cost, split by the party that pays it.

		Per step (µs)			Proof		
Corpus	Batch	Acct.	Plane	Auditor	Nodes	KiB	Indep.
64,000	64	77	250	900	583	18.2	1029
128,000	128	141	509	1796	1164	36.4	2182
256,000	255	267	1039	3597	2315	72.3	4584
1,048,576	1045	1039	4565	15028	9493	296.6	20903

Table 28 answers a question the single 29.36 ms denominator cannot: what the same added cost is worth as per-step compute moves from milliseconds to seconds. It is a *calculation* over two measurements, not a third measurement. Each cell of Table 28 divides a measured added cost per step (Table 27: record plane, prefetched, at the stated k and anchor latency) by a measured step time (`steptime.py`, CPU). The division is the calculation: it assumes the added cost does not grow with the model, which holds for the accountant and the register and only approximately for the plane, whose work grows with the batch. The ratio falls in inverse proportion to the step time, so the headline configuration costs 1.9% of the smallest measured step and 0.004% of the largest, and even the unamortised $k = 1$ configuration at a 10 ms register falls to 2.5% of a 1,501 ms step. A deployment should still measure its own ratio on its own hardware; this table says which way that measurement will move.

Table 29 gives the cost of the data plane of Section 4.2 split by the party that pays it, which the main text summarises in one line. The three roles are separate processes with separate budgets: the accountant derives a batch commitment from the beacon and tags a release token; the plane checks the token and serves the batch; the auditor verifies a Merkle multiproof against the committed root. Token authentication is a MAC rather than a signature because the token travels one hop between two parties that are both outside the trainer; Ed25519 costs 307 µs in place of the MAC's 12.9 µs and buys nothing here. Only the first two roles are on the training path. The third is deliberately not, and the reason is structural rather than a performance compromise — the dataset root is fixed before the run begins, so a batch that was substituted cannot be proved correct at any later time, and an auditor who spot-checks after the fact loses nothing by doing so. Multiproof deduplication of the shared upper frontier is what makes even the after-the-fact check affordable, roughly halving the node count against independent paths at every corpus size.

B.3 The other two preconditions: sampler and clipping

The accountant is where this paper claims novelty, but it is not the only precondition outside the measurement, and the two levers below bound what a correct accountant is worth on its own. Neither is claimed as new: the gap between a shuffling loader and Poisson accounting belongs to prior work on deployed DP-SGD implementations (Chua et al., 2024), and the machinery for comparing a declared sensitivity against a measured one belongs to the grey-box auditing line (Cebere et al., 2026).

Table 30 brackets the sampler-substitution gap. The left column is what a Poisson accountant reports; the right is the conservative endpoint in which the loader earns no subsampling amplification at all, computed exactly rather than from an Rényi bound for the reason given in Section 3.2. At 10,000 steps that endpoint is 5,426, a span of 6,888× over what is reported. A shuffling loader lies *between* the two columns, and establishing where is an open problem the honest-case literature studies and this paper does not solve; reading Table 30 as "the true ε is 5,426" is a misreading — that column is the accountant's bound at the conservative endpoint, not a measurement of the mechanism that ran. The claim is narrower: the accountant reports the left column regardless of which sampler ran, the identity of that sampler is not in the attestation report, and the interval the verifier is asked to take on faith is several orders of magnitude wide.

Table 31 makes the same argument for the clipping site, and the *order* of the operations is the whole of it. The mechanism priced there is

$$M(D) \;=\; T\Big(\textstyle\sum_{x\in D} \mathrm{clip}_C\big(g(x)\big)\Big) \;+\; \mathcal{N}\big(0,\; \sigma^2 C^2 I\big), \qquad \|T\| = L. \tag{1}$$

Table 30: The span of the sampler-substitution gap, from the Poisson-accounted value to the unamplified endpoint.

Steps	Poisson ε	Unamplified ε	Span
1,000	0.68	634	935×
2,000	0.69	1,190	1,724×
5,000	0.73	2,801	3,854×
10,000	0.79	5,426	6,888×

Table 31: Budget inflation from a transform of operator norm L applied *between* the per-example clip and the noise.

$\|T\|$	Effective σ	Claimed ε	Recomputed ε	Inflation
1.00	1.000	0.79	0.79	1.00×
1.25	0.800	0.79	1.39	1.77×
1.50	0.667	0.79	2.51	3.19×
1.75	0.571	0.79	3.94	5.01×
2.00	0.500	0.79	6.42	8.15×

Adding or removing one record moves the clipped sum by at most C and T carries that to at most LC, while the noise is calibrated to C: the effective multiplier is σ/L, and at $L = 2$ the recomputed ε is 6.42 against the claimed value in the table's first row, an inflation of 8.1×.

The order matters. If the clip is applied to the transformed gradient — the released contribution being $\mathrm{clip}_C(T(g(x)))$ — then every contribution has norm at most C by construction, the add/remove sensitivity of the sum is at most C whatever L is, and no factor arises at all. Applying T to an already noised output is a third case and changes nothing, by post-processing. Only Equation (1) is the lever, and it is the ordering Table 31 prices.

Note what is *not* broken in Equation (1): the clip is genuinely enforced, every per-example gradient really is clipped to the declared norm, and an attestation of the clipping code verifies without qualification. What the noise is calibrated against is simply not the sensitivity of what is released, and the transform lives downstream of the clip in a preconditioner or a gradient-space normalisation that no privacy review treats as privacy-relevant. This is the cheapest lever in the paper — unlike rollback it costs no extra compute at all. The grid stops at $L = 2$ because beyond it the Rényi bound underlying the recomputed column stops being tight; that column is an upper bound for Equation (1), not a minimal ε and not an audited lower bound.

C Implementation and Deployment

C.1 Implementation

The subsumed sampler and noise path. Section 3 states the two properties the accountant relies on; this is the construction. The dataset is committed once as a Merkle tree over canonicalised records, with domain-separated leaf and node prefixes and *keyed* leaves over a fixed public capacity N: a leaf is $\mathrm{HMAC}_s(\texttt{0x00} \,\|\, \langle i \rangle \,\|\, x_i)$, or $\mathrm{HMAC}_s(\texttt{0x02} \,\|\, \langle i \rangle)$ for a slot no record fills, under a uniform 256-bit per-run secret s generated inside the accountant and sealed to the plane's measurement. That is what keeps the published root from being a function of the data and the privacy experiment from collapsing; the position and the fixed capacity are both load-bearing, and Section B.1 gives the reduction. Binding is unchanged, since s is fixed before the run. The plane *does* hold s — it builds and checks the tree — and an auditor given s can open any leaf and is therefore inside the trust boundary rather than a member of the privacy adversary. The batch commitment c_i is a hash, keyed by β, of the index set selected under $s_i = \mathrm{KDF}(\beta, \kappa(j, i))$ — itself a β-keyed value, which is why the sampler hybrid H_S replaces it (Section B.1) — and the plane serves a batch only if the capability's c_i equals the commitment it derives the same way, which is what makes the *distribution* the one the amplification claims. Membership in the committed corpus is checked after the fact rather than per step: an auditor opens a Merkle multiproof for a sampled step against the dataset commitment (Table 29). Indices are drawn by exact Poisson subsampling implemented as geometric

skips $g = \lfloor \log u / \log(1-q) \rfloor$, so no index is enumerated and the cost is proportional to the batch rather than the corpus. Noise takes the whole capability — run, replica, index, batch commitment, generation and query — as its context string under a sealed seed, so no party holds a grindable seed, a re-executed step (which receives a new capability) cannot silently reuse the noise of its first execution, and a capability presented again gets its first draw back. Neither the multi-party online variant nor the offline variant is implemented or measured here.

Numerical care in the accountant. The Rényi optimum is searched over integer orders and the ends of the grid are not treated symmetrically: an optimum at the *top* is an artifact and the implementation raises, while an optimum at the bottom leaves a valid but untight upper bound — safe for a *claimed* guarantee and unsafe for an attack result, which is why the driver promotes that warning to a hard error for every recomputed-ε figure. Two further details are load-bearing and easy to miss on reimplementation. The per-step Rényi vector is evaluated in log space over the binomial expansion, since direct evaluation silently loses the large-α tail where the optimum sits at small q. The Gaussian-DP inversion is evaluated as $\exp(\varepsilon + \log \Phi(z))$, because at the unamplified endpoint the two factors of $e^{\varepsilon}\Phi(z)$ overflow and underflow while their product is bounded. The same discipline is why Table 30 is computed from the exact Gaussian-DP tradeoff (Dong et al., 2022) and inverted by bisection rather than from the Rényi accountant, whose integer-order bound is loose in the unamplified regime and would have inflated the span in our favour.

Tests. Beyond the closed-form checks of Section 3.2, stepwise composition must agree with bulk composition, and a second group tests the state machine as a state machine: an index-committing accountant detects a replayed index, a plain running total does not, restore discards accounting, and forks advance independently. Focused recovery tests additionally cover an old index first used after the last seal, a new query-bound capability at that index, immediate release and recovery, complete prospective-cost admission (base reservation plus surcharge), and injected commit failure. The bounded interleaving checker asserts both the count invariants and coordinatewise durable-cost dominance, pricing what the plane released per sample context; its advancing-noise null implementation must fail. The sample-context regressions cover a per-replica rule as a failing control, a held capability, reversed redemption, recovery between two uses of one context, and a mid-window join; the register-freshness regressions run against a replaying host, each with a control that must be fooled (A11–A16).

Reproducibility. Tables are generated, not typed. One driver writes every table file and a macro file defining each number the prose quotes, so text and tables cannot drift, and it emits a manifest recording the source revision, interpreter and library versions, platform and parameters.

How the deployed accountants were studied. The study reads the published source of seven accounting surfaces across six open-source packages and exercises each locally through its own public API, using documented entry points only; nothing is monkey-patched or written to a private attribute, no hosted service was contacted and no other party's data was touched. Every package is pinned by exact version, and each behaviour is given with file and line. The one surface that persists composition state across a restart is Opacus (Yousefpour et al., 2021); OpenDP is the one that cannot persist it at all; the remaining five offer no checkpoint path of their own and therefore no continuity check either.

Consequences of the Opacus load path. `PrivacyEngine.save_checkpoint` (`privacy_engine.py`:618) writes the accountant's state into the checkpoint and `load_checkpoint` (:659) restores it unconditionally; `IAccountant.load_state_dict` validates only that the expected keys are present and the mechanism identity matches. Because it compares nothing against live state, a load is indistinguishable from a resume regardless of whether the loaded state is older than the live one — and in the model Opacus implements there is no reason it would compare. The Merkle multiproof used by the data plane deduplicates the shared upper frontier, roughly halving node count against independent paths.

The harness, in full. Every analytic number comes from a bound, and a loose bound overstates the attack — the one direction of error this paper cannot afford — so the driver promotes an untight Rényi optimum at either end of the order grid to a hard error. The load-bearing tests are against

closed forms rather than against the implementation: at $q = 1$ the subsampled bound must collapse to $\alpha/2\sigma^2$, and T-fold composition of a μ-Gaussian-DP mechanism must be $\sqrt{T}\mu$-Gaussian-DP and strictly tighter than the Rényi bound. Records are chained by SHA-256 and heads signed in batches of k; state is sealed as one blob with the register generation *inside* it, which is what makes a stale blob detectable rather than merely old. The data plane is a third process holding the corpus and sharing an HMAC-SHA256 key with the accountant and nothing with the trainer, over a Merkle tree whose batch indices are drawn by exact Poisson subsampling via geometric skips, so no index is enumerated. The verifier is separate code sharing no state with the accountant. Each of the seven accounting surfaces was read at the source and exercised through its own public API using documented entry points only — nothing monkey-patched, no hosted service contacted, no other party's data touched.

C.2 Deployment, and where the data plane goes

Section 3 specifies the mechanism against an abstract data plane, an abstract anchor and a step that is one application of the mechanism. Real pipelines supply none of those directly. This appendix says what each becomes in a deployment, and — more usefully — what each of them costs and where the construction stops working. Nothing here is measured; it is design consequence and engineering judgement, marked as such wherever the distinction matters.

What is built, and what is drawn. One distinction runs through this appendix and is worth making once, plainly, because a reader skimming a deployment figure will otherwise take a diagram for a measurement. *Built and exercised:* the accountant state machine, the record plane, the gradient plane, the release-token path, a real external register as a separate service over a real socket, the crash/restart/rollback harness, the interleaving enumeration, and the token fuzzer. These are ordinary operating-system processes on a laptop, not confidential VMs, and every timing in this paper is a timing of *them*. *Proposed and not built:* the confidential VMs those processes stand in for, their attestation reports, key brokering on measurement, hardware sealing, and any confidential-GPU arrangement. Sealing is AES-GCM under a key we hold, which gives integrity at rest and — exactly like real sealing — no freshness, so the substitution is faithful in the property the attacks turn on. What it cannot do is substantiate a hardware-backed attestation claim or transfer a measured overhead ratio to a confidential-GPU deployment; Section C.5 states the platform requirement the design needs and which platforms currently meet it, and where this appendix reasons about a confidential deployment's cost it is reasoning, marked as such, not measurement.

In Table 32, the bold row is the configuration Proposition 1 is stated and proved over, and the others are reported as what they are. "Batches" is access accounting and "outputs" is mechanism accounting; a verifier that reads the first as the second makes exactly the substitution this paper is about. The dagger marks a counter-only register, with which the guarantee is kept and recovery from a stale blob refuses; the full register needs the unbuilt adapter of Section C.5. "Fresh" means replies bound to their requests (Section A.2).

Figure 7 is a diagram of the design, not of what ran: every measurement in this paper is a CPU prototype, and the confidential VMs, attestation reports and hardware register it draws were not exercised (Section C.2).

Where the data plane goes. The design says the data plane is the only component that can release a batch, which reads as a demand for physical sole custody of the corpus and is widely taken to rule out the object-store pipelines everyone actually runs. It is a weaker requirement than that. What gating gets its force from is not custody of the bytes but sole possession of the *capability to decrypt* them, and that separates cleanly from where the bytes sit. The corpus can be replicated across object storage, cached on local NVMe and prefetched by the trainer itself, provided every record is encrypted under a key the trainer never holds.

Three arrangements follow, in increasing order of how much of the existing pipeline survives.

In-TEE data server (proposed). In the reference deployment, the plane would be a separate confidential VM holding the corpus in private memory or on a volume whose key is sealed to the plane's measurement, and it hands batches to the trainer over an attested channel. Trust reduces to the plane's measurement and to whatever released the volume key. It is the simplest arrangement

Table 32: What each configuration binds, against which adversary; the **bold** row is the primary configuration.

Adversary	Register	Plane	Certificate binds	Faithfulness
crash, none	app. counter	record	no lost window	not bound
A1 in-image	vTPM counter†	record	batches = charged	not bound
A2 retaining	**fresh ext. CAS**	**gradient**	**distinct outputs ≤ charged**	**in-plane**
provider	independent	gradient	as above, no collusion	in-plane

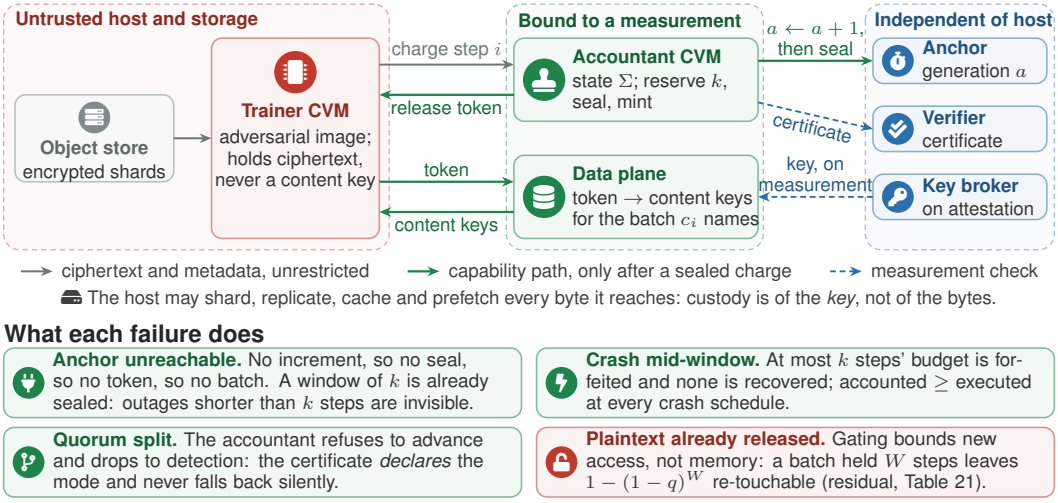


Figure 7: Where DP-ATTEST sits in a deployment.

to argue about and the least compatible with a petabyte corpus, because the plane's memory and bandwidth become the training system's.

TEE-bound storage gateway. The corpus stays in ordinary object storage, encrypted per record or per shard under content keys the gateway holds. The trainer fetches, caches and prefetches *ciphertext* on its own, which leaves the existing loader, its sharding and its bandwidth engineering untouched; the plane is reduced to a key-release service that returns the content keys for the indices a token names. Fetching ciphertext ahead of time buys the adversary nothing, so the prefetch buffer that Section 3.2 charges as a retention window shrinks to whatever plaintext the trainer is holding at once rather than whatever it has downloaded. The gateway's own key is released by an attestation-gated broker — the attested key-release primitive every confidential-computing platform now ships — so the root of trust is the same one the attestation report already rests on and no new party is introduced. The cost is one unwrap per step on the critical path and a key-management granularity choice: per-record keys make the retention window exact and multiply key state by the corpus size, per-shard keys are cheap and make W a shard rather than a batch.

Data-owner-resident plane. For cross-organisation fine-tuning the plane runs in the data owner's trust domain and streams batches to the trainer's confidential VM. The trust separation is the strongest available — the owner never exports a decryptable corpus at all — and the latency is a wide-area round trip per step, which the write-ahead window amortises for the token but not for the data.

What an adversary can still do. Take the strongest position the threat model grants: the adversary supplies the trainer image, controls the host, and has additionally compromised the storage service. Reading the stored objects yields ciphertext. Replaying an old token fails, because the plane records tokens it has served. Requesting a token for one batch and demanding another fails, because the accountant derives c_i itself from the beacon. Compelling the key broker to release the plane's key to an unattested party is possible exactly when the platform's attested key release is broken, which is the assumption the report already rests on, so the adversary must break confidential computing

as a whole rather than this mechanism alone. What remains is the inherent residual, and it is not small: plaintext that the plane *legitimately* released can be re-touched without a token for as long as the trainer keeps it. Gating bounds new access, not memory. That is the retention window W of Section 3.2, whose exposed fraction is measured in Table 22, and it is why per-record rather than per-shard keys are worth their cost in a deployment that cares.

Gradient-releasing plane. The three arrangements above all hand the trainer records, so all three bound fresh access and none bounds memory (Section 3.2). The fourth does not hand it records: the plane holds the corpus *and* applies the mechanism, and what crosses to the trainer is one clipped, summed, noised gradient. Retention then buys nothing, because there is no plaintext to retain, and the trainer is reduced to the optimiser update. The work is relocated rather than added — per-example gradients and clipping are DP-SGD's own cost — so what the arrangement charges is the difference on the wire between a gradient and a batch, which favours it exactly when the trainable parameters are smaller than the batch. That is the shape of parameter-efficient fine-tuning, and the object it moves per step is the one data-parallel training already all-reduces. What it costs instead is generality: the plane must hold the model and the framework, so it is the arrangement with the largest proposed trusted-computing boundary and the one a deployer adopts when A2 is in the threat model rather than by default.

One pipeline cannot be secured by any of this, and saying so plainly is better than a caveat: a trainer handed the plaintext corpus up front — a pre-sharded local dataset, a mounted read-only volume — has already been granted unbounded access to every record, and no discipline imposed downstream of that grant can withdraw it. DP-ATTEST is not deployable there. The obstacle is the pipeline rather than the mechanism, and it is the same obstacle that makes the ε such a pipeline publishes unenforceable by any means.

C.3 STEPS, REPLICAS, MECHANISMS AND THE SAMPLER

The unit of accounting is the *privacy* step: one application of the mechanism, meaning one noise draw against one clipped and summed gradient at one sampling rate. Almost every scaling technique in a modern trainer manipulates some other unit, and each is safe or unsafe according to whether it preserves that one.

Gradient accumulation. Accumulating over m micro-batches before a single optimiser step is one privacy step, not m: the noise is drawn once, against a sum whose sensitivity is unchanged by the order of summation. The token therefore covers the whole logical batch — c_i commits to the full beacon-derived index set. On the gradient plane, which is what is built, the plane clips and sums the whole logical batch and adds noise once, however the deployment's gradient function micro-batches internally; serving m record slices against one token, for the record plane, is a proposal we have not built. The accountant is not involved in the split either way, and a trainer that changes m mid-run changes nothing it can be charged for. Charging m steps instead would be sound but wasteful, and charging one step per micro-batch *without* drawing noise per micro-batch would be the sensitivity error of Section B.3 in a different costume.

Variable-length packing. Packing several examples into one sequence is a layout decision about the served batch, and it is safe precisely as long as the clip stays per contributing example: sensitivity is a property of what one contributor can move, not of what one tensor row contains. Since the plane serves example indices and the trainer packs them, packing changes no token and no charge. Where it does bite is on fixed-shape accelerators, which is the sampler question of Section C.3.

Data parallelism. Two configurations are *built*, distinguished by whether replicas share *sample contexts* κ (Algorithm 1). With independent contexts, $\kappa = (rid, j, i)$: replicas draw independent samples, each release on a new context is fresh, and each replica carries its own chain, head and charge. With shared contexts, $\kappa = (rid, i)$: replicas at one local index read the *same* hidden sample, and because the plane seeds noise from the whole capability, each receives its own independently noised full gradient — so the second release on a context is a repeat and is charged as one. A configuration that adds one noise draw to an assembled sum and serves the replicas slices of it, at one charge, is **not built**: the plane never slices a draw, and charging a shared release once would be an

undercount (Section A.4). The certificate is signed only over a state holding a head for every declared replica, so no configuration lets an undeclared replica contribute (Section 3.1).

Concurrency. The accountant's advance on a replica is a compare-and-swap on that replica's chain head: two submissions extending the same head serialise, the loser is offered a head that has moved, and no lock spans replicas. Advances on distinct replicas commute, because the Rényi accumulator is additive and D is per-replica.

Additivity is not a concurrency design, and we should not have implied it was. Two pieces of state are *global*, not per replica: the prepaid pool w and the budget the admission test reads. A replica cannot consume a unit or pass admission without touching both, so commuting per-replica updates do not by themselves license parallel processing. **The implementation we evaluate is serialised**: every AUTHORISE runs to completion against one accountant, and the linearisation point of an *accounting* transition is the register's compare-and-swap (Section A.2), with the in-memory decrement of w ordered by that same serialisation. Parallel processing of R replicas is therefore a *deployment proposal*, not a measured property: it needs w and the budget test made atomic — one compare-and-swap over the pair, or a short critical section around admission — and we have neither built nor measured that. The per-replica quantities (D, the chain heads, b_r) are the part that genuinely commutes. One interaction differs by plane. The *record* plane serves a token only while the register reads the generation it was minted at (its freshness floor, A9), because re-serving an older one hands the trainer a batch for a step nobody charged; so there any commit — another replica's reservation or surcharge — expires every token still held, and the holder re-authorises and pays again, which over-charges and which we have not measured under real concurrency. The *gradient* plane needs no floor — its output is a PRF of the capability, so a re-presented capability returns the bytes it returned before — and it applies none: a capability held across another replica's commit still redeems (Table 9). The anchor is the one shared resource, and it is shared far more cheaply than it first looks: the anchor orders *generations of* Σ, not steps, so a run needs one increment per window regardless of R. Duplicate handling — the repeat test *and* the first-repeat upgrade — follows the *sample context* κ, not the replica (Algorithm 1, line 12). For **independent** replicas the context carries the replica identity, so a re-executed step on replica j is priced against j's own high-water mark and two replicas at the same local index are not duplicates of each other. For **shared-context** replicas they read the *same* sample, and a second replica at an index any sharing replica has *authorised* — redeemed or still held — is a repeat: it redeems its own capability, and the capability is what seeds the noise, so what it receives is a second independently noised output over that sample rather than a slice of the first (Section A.4). The upgrade is owed once per context, whichever replica repeats first.

Stragglers and elastic scaling. The model the paper proves and the implementation runs is the one in Algorithm 1: prepaid capacity w is a *run-level pool* that any registered replica may spend, while the possible-index bound b_r is per replica and is sealed for *every* registered r (Section A.2). One consequence follows directly and is a cost, so we state it: because the pool is shared, a restart forfeits the whole outstanding reservation and therefore the prepaid capacity of *every* replica, not just the straggler's tail. A deployment that wants straggler-local forfeiture would need per-replica pools, which is a different design with a different proof; we have not built it. Adding a replica mid-run requires registering a new identity with its own chain, by a commit that seals its bound at $w - 1$ because it may spend from the open pool (Section A.2); the certificate then covers the union of declared replicas, and a replica removed mid-run contributes a final head that the certificate still names. Elasticity therefore costs a registration round trip per membership change, not per step. We have not measured a multi-replica deployment, and the claim here is about semantics rather than throughput.

Heterogeneous mechanisms. A step record already carries $(q_i, \sigma_i, C_i, m_i)$, so a schedule that changes the sampling rate, the noise multiplier or the clipping norm is representable without extension. Two things change, and the first is a constraint the recovery contract imposes rather than a presentational choice.

Why parameters are pinned, and what a phased schedule costs. Recovery reconstructs R from the register's frontier because $R = n\rho(q, \sigma) + s\Delta$ (Section A.2), and that identity holds only while every unit costs the same $\rho(q, \sigma)$ and every surcharge the same Δ. AUTHORISE therefore *refuses* a record whose mechanism is not the declared one, rather than pricing it: pricing it would make (n, s)

Table 33: Padding a Poisson batch to a fixed capacity, over a corpus of 1,000,000 records and a 10,000-step run.

z	Capacity	Wasted compute	Overflow / step	Overflow / run
3	1095	9.5%	1.4×10^{-3}	100.00%
4	1127	12.7%	3.8×10^{-5}	31.50%
5	1159	15.9%	4.2×10^{-7}	0.42%
6	1190	19.0%	2.4×10^{-9}	0.00%

an incomplete description of R and put recovery back to bounding, which Section A.2 shows cannot be made sound. A run that needs several mechanisms declares them as *phases* at registration and the frontier carries one (n, s) pair per phase — a handful of integers, and the same monotonicity argument per coordinate. A genuinely per-step σ is the case this construction does not support with an exact recovery, and the sound options there are $k = 1$ with the frontier carrying R itself, or accepting that recovery from a stale blob refuses. What changes second is how the certificate is compressed, and there are two forms with a real trade between them.

Per-mechanism counts, the form Section 3.1 describes, let the verifier recompute ε from scratch: one Rényi vector per distinct (q, σ) pair, multiplied by its count and summed. The verifier trusts the accountant for nothing arithmetical. This is efficient exactly when the number of distinct pairs is small, which is the common case — schedules have phases, not a fresh σ per step — and degenerates when a run varies σ continuously.

The alternative is to certify the accumulated Rényi vector itself. Its size is the order grid, a few hundred bytes, whatever the schedule did, and Rényi composition being additive means the accountant can maintain it at no extra cost. The verifier then recomputes only the Rényi-to-(ε, δ) conversion and takes the per-step arithmetic on the accountant's measurement. Generality is bought with checkability, and since both forms cost the accountant nothing, the right behaviour is to emit both and let the relying party decide which it is willing to lean on.

Reservation against a moving schedule. Write-ahead reservation charges a window before the window runs, which requires knowing what the window will cost. A schedule declared at registration poses no problem: the window's charge is read off the schedule and there is nothing to reconcile. A schedule that is genuinely adaptive does, and the rule is to charge the window at the most expensive parameters admissible within it. Steps that then execute more cheaply are over-charged, in the permitted direction, and the waste is proportional to the spread of the window's admissible parameters — a third reason to keep k small. One caveat is easy to miss: a σ chosen by looking at the data is itself a data-dependent release, and it must be accounted as one. DP-ATTEST does not do that for the deployer, and a run whose schedule adapts to a validation metric computed on the private corpus has a privacy cost this design binds no better than any other.

Which sampler actually ran. Poisson subsampling is not a preference of this design; it is what the amplification theorem quantifies over, and a deployment that shuffles instead is claiming an ε its loader did not earn — prior work's finding (Chua et al., 2024), bracketed in Table 30, not ours. The question a gating data plane raises is narrower: can the plane run the sampler the accounting assumes, at scale, without paying for it on the critical path?

For the draw itself the answer is yes, and for a reason peculiar to this design. Because indices come from a beacon settled before the run, the entire schedule is determined at step 0, so batches can be derived arbitrarily far ahead of the step that consumes them; the geometric-skip draw is $O(\text{batch})$ rather than $O(\text{corpus})$, and it runs off the critical path in either case. The sampler is not the cost.

The variable batch *size* is. Fixed-shape accelerators want a constant token count, Poisson does not supply one, and the tempting fix — draw Poisson and drop the overflow — changes the sampling law and voids the amplification the accountant is claiming, which is precisely the class of error this paper is about. Padding does not: sample at rate q, pad to a fixed capacity with records that contribute nothing to the gradient, and account at q. Table 33 prices it. At the paper's q over a million-record corpus, a capacity of 1159 — 16% wasted compute — leaves a 0.4% chance that any step in a

10,000-step run overflows, and an overflow is handled by discarding the step and charging it anyway, in the permitted direction.

The more interesting accommodation is to concede the shuffling loader rather than fight it. DP-ATTEST can serve a shuffle-derived batch perfectly well; what it will not do is let the run *claim* amplification for a sampler it did not use, because m_i names the sampler in every record and the certificate carries it. A shuffling deployment under DP-ATTEST is accounted at a larger ε it has earned, where today it is accounted at a smaller one it has not. That is a utility loss measured against a number that was never real.

C.4 WHEN THE ANCHOR FAILS

The anchor is a safety device and must not become a liveness device by accident. Its unavailability stops the run rather than weakening it: no increment means no seal, no seal means no token, and no token means the plane serves nothing. That is the fail-closed direction, and the write-ahead window is what makes it tolerable rather than crippling — a window of k steps is already sealed before its first token is minted, so an anchor outage shorter than k step times is invisible to the trainer. Choosing k therefore trades three things at once and not two: forfeiture on crash, overhead per step (Figure 3(b)), and tolerance of anchor unavailability. What it does not trade is safety, and Table 34 is the reason the third of those can be bought freely: the write-ahead ratio is flat in k, so a deployer sizing k for anchor availability is spending utility and nothing else.

In Table 34, each design is reported under both crash strategies the sweep runs, which is the same decomposition Figure 2 plots as separate curves. A record the accountant refuses receives no token and therefore no batch, so it is an attempt and not an execution.

Partition. Where the anchor is a replicated service, an accountant that cannot reach a quorum must refuse to advance. It may serve the remainder of the window it has already sealed and must then stop. The cost of a partition is thus bounded by the outage plus at most k forfeited steps, and it is a denial of budget rather than a disclosure.

Inconsistent quorums. Two quorums each believing they hold the counter is the split-brain that the majority-intersection assumption of the state-continuity literature (Matetic et al., 2017; Angel et al., 2023) exists to exclude. Where it cannot be excluded, DP-ATTEST degrades from prevention to detection: the certificate names the anchor generation it was signed against, so a verifier holding two certificates for one run at the same generation with different chain heads has caught the fork after the fact. This is strictly weaker, and the important engineering rule is that the downgrade must be *declared*. The certificate carries the mode, a verifier can refuse a detection-mode certificate for a release it considers high-stakes, and a silent fallback — an accountant that quietly keeps going when its anchor stops answering — would be worse than having no anchor at all, because it would produce a certificate that looks like the strong one.

Why not prove it instead. Section 5 gives the structural reason a proof cannot stand in for the anchor; the operational shape of the alternatives is worth setting out, because the choice is real even though the substitution is not. A recursive proof chain does bind a step count, but only relative to whatever remembers the latest head, so it relocates the freshness requirement into the verifier's memory rather than removing it — an anchor under another name, and one that must be online at verification time rather than at training time. A public ledger is an anchor with a different availability and privacy profile: stronger against a colluding operator, worse for a deployment that would rather not publish the existence and cadence of its training runs. Against those, the counter we assume is the cheapest object that does the job, and the write-ahead window is what makes its cost per step negligible: one increment per k steps, on a path that a proof system would occupy per step. The right reading is that DP-ATTEST needs *some* monotone object outside the prover, and names the weakest one that suffices.

Partial commits. The commit order leaves exactly one window open, between the increment and the persist. An anchor ahead of the sealed state is reconciled from the register's *frontier*, not from the size of the gap: recovery burns every replica's mark to $\max(b_r, B^\star)$ and restores (n, s) from $(n^\star, s^\star)$, which restores R exactly (Algorithm 2). The stale accumulator is never reinstated.

Table 34: Steps *served* per step accounted, across commit intervals; above 1 the accountant undercharges.

	Write-ahead		Seal every k		Worst schedule, seal every k		
k	crash	+rollback	crash	+rollback	m	Served	Accounted
50	1.00	1.00	40.20	40.20	49	2,010	50
100	1.00	1.00	40.60	40.60	99	4,060	100
250	1.00	1.00	40.84	40.84	249	10,210	250
500	1.00	1.00	40.92	40.92	499	20,460	500
1000	1.00	1.00	40.96	40.96	999	40,960	1,000
2000	1.00	1.00	40.98	40.98	1999	81,960	2,000

A tempting rule that is unsound. Charging $g \cdot k$ steps for a gap of g generations is **unsound**: recovery itself raises the high-water mark, so the next reservation sets b_r from the raised value and nothing linear in g is closed under RESTART $\rightarrow$ AUTHORISE $\rightarrow$ RESTART (Section A.2 gives the composition that breaks it).

C.5 CONFIDENTIAL HARDWARE

Section 5 states the gap; this is what closing it involves.

The accountant is the easy part. It is a small CPU-only service that runs unmodified in an SEV-SNP or TDX guest; the report fetch is once per run, and sealing uses a guest-unique key derived through the platform's key-derivation request. Neither is on the per-step path, so the overhead of Figure 3(b) is not expected to move.

The anchor is the part the platforms do not help with, and this is worth being precise about, because "no confidential VM ships a counter" is too quick: the platforms ship several things that look like one.

TDX runtime measurement registers are extend-only, which gives an append-only log *within* a boot and exactly nothing across one, since they reset when the trust domain launches — and relaunch is the adversary's move. The guest vTPM that Azure and GCP confidential VMs expose does provide TPM 2.0 NV monotonic counters that survive a reboot, and it is the closest thing available; what it does not provide is resistance to the party running the VM, because the vTPM's state is persisted and served by the host or its paravisor. That distinction is exactly the A1/A2 split of Section 2 applied to the platform rather than to the trainer, and it decides *who* may operate the register, not whether a counter *is* one. Against this paper's *primary* adversary — a compromised dependency inside an otherwise honest deployment, where the platform operator is not the attacker — the guest vTPM is an acceptable *place* for the register, since the party able to rewind it is trusted; but its NV counter alone is **not** the register the protocol needs, because it carries no payload (below). What it supplies without further engineering is a counter-only register: the guarantee is kept and recovery from a stale blob refuses. The full register — generation, digest and frontier advanced as one monotone, fresh transition — needs the adapter described next, which we have not built. Against the secondary adversary, the provider itself, even that placement fails, and the externalised register of Section 3.2 is required. A deployer should choose the register by the adversary it is defending against, and the certificate should name which was used.

Two further properties are asked of the anchor, and they are cheaper than they sound. It must hold a payload — a 32-byte lineage digest and the three integers of the accounting frontier — and it must answer the generation and that payload as *one* authenticated snapshot (Section A.2). The quorum service of Section 3.2 returns the tuple in a single round trip per replica and is what we exercised.

A TPM NV index is not this object by default. Holding data and being read in one operation do not make it the register. TPM 2.0 distinguishes an ordinary writable NV index from an eight-octet counter index advanced by `NV_Increment`: the first stores a payload but is freely rewritable, so it supplies no monotonicity; the second is monotone but carries a counter, not a digest and three integers. Neither is an atomic monotone *tuple*-compare-and-swap. A realisation needs an adapter that makes the pair behave as one transition — for instance a counter index for a beside a policy-protected data index for (d, F), with the data write authorised by a policy bound to the counter's next value, so the payload cannot be written except in step with the increment, plus the freshness binding of

Table 35: Platform facilities, and which could serve as the anchor.

Facility	Platform	Survives relaunch	Host-rewindable
Launch measurement	SNP, TDX	n/a	—
RTMR extend	TDX	no	—
Sealing, key deriv.	SNP, TDX	yes	*yes*
vTPM NV counter	SNP, TDX	yes	*yes*
SGX monot. counter	SGX only	yes	no

Section A.2. We have not built or measured that adapter, and we no longer imply the platform supplies the contract for free; what we claim is that the contract is implementable on such a platform, not that NV storage is already it. A counter-only register — SGX's, or a bare hardware monotonic counter — keeps the guarantee and loses the ability to continue past a rollback: RESTART refuses rather than reconstructing (Algorithm 2, line 11). That is the right way round, and it should be a deployment decision rather than a surprise.

Against the provider: a concrete externalised register. When the party the guarantee is claimed against controls the platform, the register must live outside it, and the rollback-protection services this paper consumes already provide the object; we describe the mapping onto one, NIMBLE's replicated ledger (Angel et al., 2023), as a design we have not built. A per-tenant append-only log whose appends are conditional on an expected height, whose read of the latest entry is bound to a client nonce, and whose endorsers are TEE processes replicated so that no single host can rewind or fork the log is a tuple register once the fields are assigned: the generation a is the log's height; the payload (d, F) is the appended entry; CAS from a is a conditional append at height $a + 1$, which loses exactly when another holder advanced first; the snapshot read of Algorithm 2 is the nonce-bound read of the latest entry, which is the request-bound freshness of Section A.2; and monotonicity of F is checked by the accountant before each append and is auditable afterwards, because an append-only history cannot hide an entry that regressed. The deployment decision that remains is who hosts the endorsers: against the provider, a majority must run on hardware it does not control — the customer's, or an independent party's — which is the majority-intersection assumption this line of work already makes (Section C.4). Its cost is one conditional append per reserved window and one nonce-bound read per restart, a quorum round trip in the regime Figure 3(b) prices.

What the prototype exercises, and what it assumes. The distinction matters because the register's properties are the ones the proof leans on hardest. **Exercised**: the atomic snapshot and the monotone compare-and-swap, in both registers — in process as `LocalAnchor` and over sockets as `QuorumAnchor`, one `SNAP` round trip per replica with the quorum rule applied to the tuple — together with refusal on a rewound or counter-only register (Table 24, A10); and freshness — authenticated replies bound to a nonce and to the exact request — on both the socket path the timings use and the in-process client, against a host that records, withholds and replays authentic replies, including the plane's own reads (A11–A16, Section A.2). **Not exercised**: for a vTPM, that an NV counter plus a policy-bound data index delivers the tuple transition across a reboot on a real confidential VM, and that the host serving the vTPM cannot roll it back. No TPM, vTPM or confidential VM was run for any number in this paper (Section C.2); the socket quorum service stands in for the register, and the adapter above is a proposal.

The data plane against a confidential GPU is where the engineering sits. In current confidential GPU modes the PCIe path is encrypted and all host–device transfer passes through a bounce buffer in shared memory that the CPU TEE encrypts and decrypts. Three consequences follow. The plane's release path terminates in the CPU TEE rather than on the device, so the GPU never verifies a token and needs no changes; a full deployment verifies two attestation reports, the CPU guest's and the device's, and the certificate should name both measurements rather than pretending one covers the other. The bounce buffer raises the cost of moving a batch, which lengthens the step and therefore *lowers* the ratio Figure 3(b) reports — the direction is favourable, and a deployer should still measure rather than transfer the number. And a dataset cache resident in device memory is a retention window in a new place, governed by the same W and the same $1 - (1 - q)^W$, which is an argument for sizing device-side caches as a privacy parameter and not only as a performance one. The crash-schedule

guarantee is a property of the commit protocol rather than of the hardware and is unaffected. None of this is measured.

D Ethics and Open Science

D.1 Ethical considerations

No third-party system was attacked. Every experiment in this paper runs on infrastructure we control. The study in Section 4 reads the published source of seven accounting surfaces across six open-source packages and exercises them locally through their own public APIs. No hosted service was probed, no account was created, no rate limit was tested, and no other user's data was touched at any point.

No personal data. The training data is either synthetic, generated by the artifact itself from a fixed seed, or the public UCI optical-digits set, whose handwritten-digit bitmaps carry no identifying information. The canary records used to reason about privacy loss are our own constructions. No human-subjects data is involved.

Disclosure. The behaviour we report is not a memory-safety flaw or a broken cryptographic primitive; it is an absent check in a component whose threat model never included a rewinding caller. The paper therefore names exact versions and does not call the absence of an unpromised rollback guarantee a package defect. The concrete mitigation in Section 3 is included with the attack.

Dual use. The attacks are directly usable by a provider wishing to overstate a privacy guarantee, and we considered withholding the end-to-end demonstration. We publish for three reasons. The capability is not created by this paper: the checkpoint API is documented, the absent check is visible in a few lines of published source, and any operator who has recovered from a preemption has already executed the mechanism without noticing. The party who benefits from secrecy here is the party misreporting, not the data subject. And the defence travels with the attack in the same artifact. The asymmetry favours disclosure: a data subject cannot audit a privacy claim they do not know is unenforced, while an operator can fix a load path in an afternoon.

Who bears the risk. The subjects of a covertly weakened guarantee are people whose data was used in training and who cannot observe any of this. They are not parties to the transaction and cannot inspect the accountant, so the harm from silence falls on them and the harm from publication falls on providers who would rather it stayed quiet. We think that settles the balance, and we state it explicitly rather than leaving it implied.

D.2 Open science

Everything needed to reproduce the results is released, and no result depends on hardware, data or software that a reader cannot obtain.

Artifacts and availability. An anonymous supplementary archive accompanies this submission. It contains the prototype (the accountant, DP-ATTEST and the data plane), every attack and experiment, the test suite, the manifest and cost tools, the pinned environment, and the recorded result records, each carrying a provenance block; its README gives one command per result family and names the record that command writes. The analytic attacks of Sections B.2 and B.3 are closed-form computations (`experiments.py`); the measured and trained results each have their own driver; and the one dataset ships inside scikit-learn, so nothing is downloaded. The archive contains no version-control history, tracking or analytics, and it contains everything: no dataset is under licence restriction and no component is proprietary. On acceptance the code becomes public under a permissive licence at a stable, non-anonymous URL.

Reproducing the results. Every number is CPU-only, and `artifact_cost.py` reports wall-clock and peak resident memory per target so a reader can plan rather than guess; it writes every target's record to a scratch path, so running it never overwrites a reported result. On the reference

machine, at host load up to 28, the accounting tests take 3 s at 207 MB, the retention and gradient-plane measurements 19 s, the quick anchor suite 2 s and the smoke checker 81 s; all stay inside 542 MB. The interleaving check has two configurations and the artifact reports both, because a cost quoted without one is unreadable: the *smoke* configuration is depth 4 over the 7-verb alphabet and depth 3 over the stale alphabet, on both planes with every named trace, which is what `artifact_cost.py` times; the *reported* configuration is the depth-6 run over both planes behind Section 4.3, at 235,298 sequences. Both use the same alphabet — authorise, serve, replay, crash accountant, crash plane, roll back store, roll back plane — and the same $A \geq E$, $A \geq M$, and coordinatewise durable-cost invariants; only depth and the plane differ. The targeted old-index/new-query recovery trace and its cost-admission and failed-commit null cases run beside both configurations. A second alphabet drives a second campaign: the *stale-state* configuration runs depth-5 over 9 verbs — the seven above plus *re-authorise* and *roll back deep* — on both planes, 118,098 sequences, and without those two verbs no ordering can produce a surcharge-only commit or reach a blob older than the previous generation. The two stale-blob schedules of Section A.2 run by name beside it with the released-object costs and the used-sample set held outside the accountant, together with the latest-blob positive control that keeps the frontier from taxing ordinary crashes. A third campaign is of a different kind: the *composed* schedules interleave a recovery with a later reservation and a second recovery, which no enumeration over a fixed alphabet reaches at a tractable depth because it needs a specific generation restored twice with work in between. The shared-context traces (Table 9) and freshness positions (A11–A16) also run by name, in seconds. The composed schedules run by name, and each emits a per-event implementation log — generation, register frontier, sealed generation, high-water mark, each re-offered index's actual disposition, the durable and required order-two costs, and the used-sample set — so the claim is a trace a reader can read rather than a pass/fail. The two long targets are the membership audits and the 40-seed training sweep, which train many small models and take tens of minutes. Nothing requires a GPU, a confidential VM, or any hardware we have and a reader does not — which is also this artifact's ceiling, since DP-ATTEST has not been run on the hardware whose guarantees it is designed around. The paper's tables and the numbers its prose quotes are generated from these records rather than typed, so text and records cannot drift apart. The library study pins every package by exact version in `requirements.txt`; it reads library source and public APIs and contacts no network service. Each run writes a manifest recording interpreter version, package versions, platform and parameters alongside its results.

Which record backs each float. A number is only as reproducible as the record behind it: two result files written by different revisions of one sweep function can disagree at the same (k, m) with no disagreement about the system. Table 36 names the record and the script behind each float, and we ship a manifest that can be checked rather than a promise.

`manifest.py` writes that manifest, `eval/results/MANIFEST.json`, carrying the source revision and whether the tree was clean, interpreter and library versions, the SHA-256 of every generated result *and* every source file, the modes each number was produced under — read off the constructor's own defaults rather than typed, so the manifest cannot drift from the code — the ordered commands that regenerate each principal trace, every named regression, and each checker campaign's depth, size and violations. It also records a *repair linkage*: `state_machine.json` records the SHA-256 of the sources the checker *imported*, and the manifest compares them with the sources shipped beside it, so results produced by a different revision than the archive's show as a mismatch rather than as a rounded number that happens to agree. `defense.json` is the one result file whose timing blocks are not regenerated, because they must not be re-measured on a loaded host; `manifest.py --reverify` re-runs its five attack rows and its crash sweep against the current source and records that every non-timing field is identical to nine decimals. A reader can therefore check the correspondence between this PDF, the archive and the results without taking any of it on trust.

Figure 2 and Table 34 call the *same* function, `defense_eval.attack_crash_aligned`, over the same schedule grid; the figure's plotted peak at $m = k - 1$ and the table's $k = 500$ row are therefore the same measurement by construction, and the figure generator asserts that the two write-ahead series coincide rather than drawing one line for two. The table generator refuses to emit a Table 4 row whose defended recomputation is missing, instead of falling back to the undefended one.

Event vocabulary. The terms below are not interchangeable, and the columns of Table 7 are named in them.

Table 36: The record and the script behind each float.

Float	Record	Script
Tables 15, 16, 18, 19, 30 and 31	`attacks.json`	`experiments.py`
Table 5	`library_study.json`	`library_study.py`
Tables 17 and 25	`production_accountants.json`, `resume.json`	`production_accountants.py`, `resume_experiment.py`
Tables 4, 21, 27 and 29, Figure 2	`defense.json`	`defense_eval.py`
Table 34	`crash_k.json`	`crash_k_sweep.py`
Tables 23 and 26, Figure 6	`audit*.json`, `release_model.json`	`audit.py`, `release_model.py`
Table 22	`retention.json`	`retention.py`
Table 24	`anchor_adversary.json`	`anchor_adversary.py`
Tables 6 to 8, Figure 3	`real_training.json`	`real_training.py`
Section 4.3, Tables 9 and 10	`state_machine.json`	`state_machine.py`
Table 12	`sensitivity.json`	`sensitivity.py`
overhead denominators, Table 28	`steptime.json`	`steptime.py`

- An **authorisation attempt** is a record offered to the accountant. It may be refused — wrong head, non-contiguous index, undeclared replica, budget exhausted, mechanism not the declared one — in which case no capability exists and no data moves.
- A **minted capability** is an attempt the accountant charged and issued a token for. It is an upper bound on releases and not equal to them: a trainer can hold a token it never redeems, which is why Table 7 reports 72 minted against 71 released in one crash row rather than rounding the two together.
- A **release** is a capability the plane actually redeemed, and a **distinct output** is a release whose bytes differ from every earlier one. Repeated presentation of one capability yields one distinct output.
- A **sample context** is the hidden sample a release reads, $\kappa(j, i)$. Several distinct outputs can share one — a re-execution, or two replicas sharing contexts — and the strict price is set per context, so this is the fourth count and the one that decides the charge. Tables 9 and 10 report all four side by side.

Which term each claim uses matters. Proposition 1's M is *distinct outputs*, and the mechanism-completeness experiments measure actual releases, because that is the quantity the privacy statement is about. *Served*, wherever a ratio in Table 34 or Figure 2 uses it, means an actual release; the conservative authorisation count is reported beside it and is the larger number. **Accounted** is the accountant's durable step count, and a ratio above one is privacy that was spent and never charged. **Recomputed** ε is always the recomputation of the trace *that column's own system* ran, with a sample released T times priced at $T\rho(1, \sigma)$ as the retry contract requires (Section B.1); the two halves of a Table 4 row therefore describe two different executions and are not a difference.

One transition, one implementation, one trace. The manuscript states a protocol, proves a property of it, and reports that an implementation was checked against schedules. A pass count does not establish that the transition a theorem needs is the transition the code performs, so this subsection maps the three onto one another. Every row names a transition of Algorithm 1 or Algorithm 2, the function that implements it, and the trace or regression that exercises it.

- **Reservation (Algorithm 1, line 17):** R, n, s, w, b_r, U **move together**
  code: `_reserve`, `_delta_units`
  evidence: `test_the_surcharge_counter_tracks_every_delta_charged`, both branches
- **Complete prospective cost (line 15), upgrade included**
  code: the `take` loop in `_reserve`
  evidence: `test_the_first_repeat_upgrade_enters_the_admission_check`

- **The gate as a Rényi filter: one declared order, one budget, halt before it is crossed**
  code: `filter_order`, `_gate_eps`, `_gate_orders`; `verify_certificate` recomputes at the same order
  evidence: `test_an_adaptively_stopped_run_stays_inside_its_pinned_budget`, whose length its own refusals decide
- **Repeat test by sample context, over authorisations (line 12):** $d_i = \mathbf{1}[i \leq \max_{r \in G(j)} D.\text{high}[r]]$
  code: `_context`, `_context_seen`
  evidence: `shared_context_traces` (Table 9), whose per-replica control fails; `test_a_per_replica_repeat_rule_undercharges_a_shared_context`; `test_a_held_capability_makes_the_next_one_on_its_context_a_repeat`
- **Strict upgrade (line 12):** $m_i \in \{0, 1, 2\}$**, once per *context*, recorded in** U **before the seal that pays it**
  code: `upgrade` in `authorise`; the `upgraded` set of contexts
  evidence: `contract_traces.first_repeat` — per-event s and identity; `test_the_upgrade_is_owed_once_per_context_not_per_replica`
- **Elastic join mid-window: the joiner's bound is sealed at** $w - 1$
  code: `register_replica`
  evidence: `test_a_replica_joining_mid_window_cannot_reopen_its_indices`
- **Surcharge-only commit (the** $m_i > 0$ **branch)**
  code: the `surcharged` path in `authorise`
  evidence: the same trace, its second and third events
- **Pooled reservation, per-replica range (line 17):** w **is run-level,** b_r **is sealed for *every* registered** r
  code: `_reserve`: `_reserved` scalar, `_reserved_high` over `self.replicas`; `_load` burns every replica on both paths
  evidence: `test_a_pooled_reservation_seals_every_replicas_index_range`; `mixed_replica_traces` with its failing control (Section A.4)
- **Availability: ordinary completion, clean recovery and a shortened final window each *progress*, not merely stay safe**
  code: the `max` at line 17; `restart`
  evidence: `mixed_replica_traces.availability` (three cases, each recording progressed-or-refused); `test_a_shortened_final_window_does_not_lower_a_sealed_bound`
- **Recovery commits on the reconstructing path only (line 14), not on the exact path (line 10)**
  code: `restart`: `_commit` after `_load(frontier=\ldots)`; the clean branch returns first
  evidence: `composed_recovery_traces`, whose generation column shows the asymmetry
- **Latch on *any* failed durable write (lines 3, 4)**
  code: `_commit`'s `except`; `_failed_closed`
  evidence: `test_a_refused_cas_latches_as_a_failed_seal_does`, over refused-CAS, suppressed-advance and failed-seal
- **One authenticated register snapshot (Algorithm 2, line 2)**
  code: `_register_snapshot`; `snapshot` on both registers
  evidence: `test_the_register_answers_one_atomic_snapshot`
- **Fresh, request-bound register replies; nonce from outside restorable state (Section A.2)**
  code: `RegisterService`, `FreshRegisterClient`; attacked through `ReplayingProxy`
  evidence: Table 24 A11–A16, each with a control that is fooled; `test_a_replayed_register_read_cannot_make_a_stale_blob_look_current` and three siblings

- **Digest taken before mutation (line 3)**
  code: `_blob_is_current`, over the sealed dict
  evidence: `test_the_named_state_digest_is_compared_before_w_is_zeroed`
- **Fetched state verified, not trusted (line 7)**
  code: `_fetch_named` and the recheck in `restart`
  evidence: `test_a_fetched_state_is_verified_against_the_register`
- **Refuse without a frontier (line 11)**
  code: `restart`'s `frontier is None` branch
  evidence: `test_recovery_refuses_when_the_register_carries_no_frontier`; Table 24 position A10
- **Reconstruction (line 13)**
  code: `_load(frontier=\ldots)`
  evidence: `contract_traces.upgrade_after_stale_recovery` and the composed schedules
- U **lost to a rollback: the upgrade is paid again**
  code: `upgraded` restored from the loaded blob
  evidence: the same trace, its last two events
- **Fresh-once: a used context refused, not priced (line 13)**
  code: `authorise`'s `repeat_policy == "refuse"` branch; the policy sealed in Σ and checked by `_load`
  evidence: `test_fresh_once_refuses_a_repeat_after_a_stale_rollback` and five more; `fresh_once_campaign` with its priced control
- **Ordered release log: the corollary's hypothesis, recorded; enforced instead, under fresh-once**
  code: `DataPlane.release_log`, `non_interleaved`
  evidence: last column of Table 9; `mixed_replica.implementation.release_order`

Which mode each result family runs in. Five settings change what a number means, and a reader should not have to infer any from its size. The *repeat policy* selects whether a repeated context is priced (the default) or refused (fresh-once); the *retry contract* selects what a priced repeat is charged; the *filter order* selects whether the budget is enforced at one declared Rényi order or at the grid minimum (Section B.1); the *replica mode* selects whether replicas share sample contexts; and whether the run's *release order is non-interleaved* decides whether Corollary 1 applies to it at all or only Proposition 1 does. Every number in this paper is produced under one of the following, and nothing is blended across rows.

- **Table 19 is the one exception to "strict everywhere"** and is listed first so the rule below is read with it. It prices every touch at the declared (q, σ), re-executions included, because it is a closed-form comparator model that never runs the accountant.
- **Fixed-schedule accounting and utility** — Table 6, Table 4, Table 32, Table 27, Table 29, the k curve, the crash sweeps, and every checker campaign: `retry_contract="strict"`, `filter_order=None`. The schedule, its crash points and its parameters are fixed before the run, so the vector is a deterministic function of declared quantities and the grid minimum is sound. *Release order:* the training results (Tables 6 to 8) resume past the high-water mark on every restart, so no context repeats and the order is non-interleaved trivially — Corollary 1 applies, under its assumptions. The defended rows of Table 4 other than replay likewise repeat no context. The **replay** row re-executes 1,000 contexts after later ones, and the checker's *re-authorise* campaigns, Table 10 and the stale-recovery row of Table 9 interleave by construction: for those, the strict charge is what the accountant pays and Proposition 1 is what they demonstrate; the corollary does not certify it as a privacy bound.
- **Repeat policy.** Every result above runs the default, priced repeats; the training runs never repeat a context, so they meet the corollary's hypothesis without the certificate enforcing it. Fresh-once is exercised by the checker's fresh-once campaign and its tests, where the hypothesis is enforced and the certificate says so (Corollary 2).

- **Replica mode.** Single-replica results are unaffected by it. The multi-replica traces state theirs per row (Table 9); the mixed-replica ownership trace runs with independent contexts.
- **Diagnostic replay rows** — Table 19 does *not* run the accountant at all: it is a closed-form comparator model in `experiments.py`, and its recomputed column prices every touch at the declared (q, σ). No retry contract applies to it. The appendix reports what the strict contract would charge for the same trace beside it (Section B.2). The surcharge *regressions* do run the accountant, under `strict` with `filter_order=None`.
- **Adaptive mode** — `test_an_adaptively_stopped_run_stays_inside_its_pinned_budget` alone: `filter_order=14`, with both the index choice and the run's length derived from the released outputs. *No table in this paper is produced in this mode.* It exercises the pinned-order *mechanism* and nothing more — **there is no adaptive privacy corollary for it to be scoped to**, because the per-round bound such a corollary would need is false (Section B.1).
- **`amplified-first`** is registered, checked and available, is used by *no* reported number, and is **outside the defended privacy claim**. Its proven domain is exactly this: the closed-form inequality at *order two* for a single reused sample under one (q, σ, T), verified numerically over the budget's range at registration. That is a check on one coordinate of one construction, not a theorem over arbitrary datasets, transcript dependencies or the order grid, and we do not offer it as one. A deployment selecting it is selecting an experimental option.

Two kinds of evidence, which are not interchangeable. A regression and a theorem answer different questions, and we keep them apart.

Operational enforcement, established by running code. That the gate refuses before the budget counting the complete price; that a reservation seals every replica's range; that recovery reconstructs and re-derives b; that a pinned order is what the gate tests, reports and certifies. These are properties of an implementation, and the regressions and traces named above are the right evidence for them. Proposition 1 is in this category too: it is a statement about bookkeeping integrity over reachable states.

Distributional correctness, established only by proof. That a given charge upper-bounds a Rényi divergence. No run of the state machine can establish this, and a run that stays under a threshold is not evidence for it. The per-round bound of Section B.1 is the case in point: the adaptive regression passes, yet the bound it might be taken to support is false. Accordingly, the strict contract's group charge is proved (quasi-convexity over the unconditioned transcript), and the adaptive extension is *not* claimed.

Two things that map holds fixed. The identity $R = n\rho(q, \sigma) + s\Delta$ is asserted *after every event* in the contract traces, not once at the end, because it is the premise Algorithm 2 reconstructs from and a transition that breaks it mid-schedule would be invisible to an end-state check. And the default retry contract is the *strict* one throughout — the traces, the regressions and every generated table — so a reader never has to decide which pricing a number came from.

Third-party software under study. The findings in Section 4 concern specific released versions of six open-source packages, all installable from public package indexes and all pinned in the artifact. We give the file and line of each behaviour we describe so a reader can confirm it by inspection without running anything.