Custodian: Bounding Access-Pattern Leakage in Encrypted Multi-Tenant Key-Value Stores

Anonymous Author(s)
Submission Id: 1904

Abstract

Encrypted key-value stores promise confidentiality for multi-tenant cloud deployments, yet the access patterns and response volumes they reveal during normal operation have repeatedly been shown to leak enough structure for an honest-but-curious operator to reconstruct plaintext ranges and query distributions. We present Custodian, a gateway that sits between tenants and an encrypted backend and bounds two families of leakage — access-pattern leakage and volume leakage — without modifying the backend's storage format or requiring a trusted execution environment. Custodian batches requests into fixed-width epochs, pads response volumes to a small set of size buckets chosen from the tenant's historical query distribution, and applies a lightweight oblivious shuffle to reorder same-epoch reads before they reach the backend. We formalize the resulting leakage as a bounded mutual-information channel and show that Custodian's reconstruction-attack advantage falls below that of three deployed baselines — Nimbus KV, Apex Store, and Vertex Store — while adding 9–18% throughput overhead depending on the configured epoch width. We further show that padding alone, without epoch-level shuffling, leaves an attacker with a residual advantage nearly triple Custodian's, indicating that padding and reordering are complementary rather than substitutable defenses.

CCS Concepts

• **Security and privacy → Management and querying of encrypted data**; *Formal security models*; *Database and storage security*.

Keywords

encrypted databases, access-pattern leakage, volume leakage, leakage-abuse attacks, searchable encryption, oblivious algorithms

1 Introduction

Multi-tenant cloud key-value stores are increasingly deployed with per-record encryption: every value is sealed under AES-GCM (or an equivalent authenticated cipher) before it leaves the tenant's control plane, and the cloud operator never holds a key. This is enough to defeat an adversary who only inspects ciphertext at rest. It is not enough to defeat an adversary who watches the store *operate*. A cloud operator, a co-located tenant with a side channel, or a subpoenaed log pipeline can all observe which encrypted record identifiers are touched by a query and how many bytes come back, even when every byte of payload is opaque. A decade of leakage-abuse research has shown that this residual signal — access pattern and response volume — is often sufficient to reconstruct the plaintext key distribution and, in structured workloads, individual record values [1–3].

Two properties make this leakage practical to exploit rather than a theoretical curiosity. First, real workloads are skewed: a small number of hot keys (billing accounts, feature flags, session tokens) dominate query volume, so an adversary with even coarse auxiliary knowledge of the key distribution — a public schema, a competitor's marketing page, a leaked support ticket — can align observed frequencies to plaintext identities with surprisingly high accuracy. Second, most deployed encrypted stores return responses whose size is a deterministic function of the matched record set. Order-preserving and order-revealing schemes, still common in commercial offerings for their query performance, additionally leak the *relative order* of keys, which combined with volume leakage enables near-complete range reconstruction from a modest number of observed queries [1, 3].

Existing countermeasures sit at two extremes. Oblivious RAM (ORAM) constructions eliminate access-pattern leakage asymptotically but impose poly-logarithmic bandwidth blowup and typically require restructuring the storage layer itself, which is a nonstarter for operators who cannot replace a production key-value store [5]. At the other extreme, response padding alone is cheap to deploy but, as we show in §5, leaves an adversary with a residual reconstruction advantage nearly triple that of a defense that also decorrelates request timing and order.

This paper presents Custodian, a leakage-bounding gateway that sits between tenants and an unmodified encrypted key-value backend. Custodian requires no change to the backend's storage format, no trusted execution hardware, and no client-side re-encryption scheme; it is deployed purely as a proxy. It combines three mechanisms — epoch batching, volume padding against a per-tenant bucket set, and an oblivious intra-epoch shuffle — that are individually cheap and, as we argue formally in §4, jointly close a leakage channel that neither mechanism closes alone.

Contributions.

(1) A gateway architecture, Custodian, that bounds access-pattern and volume leakage for existing encrypted key-value stores without modifying the backend (§3).
(2) A mutual-information formalization of the residual leakage channel that Custodian exposes, together with a closed-form bound relating epoch width, bucket count, and shuffle entropy (§4).
(3) An evaluation against three deployed baselines — Nimbus KV, Apex Store, and Vertex Store — on three synthetic multi-tenant workloads, showing that Custodian reduces reconstruction-attack accuracy by 31–54 percentage points relative to the strongest unmodified baseline at 9–18% throughput overhead (§5).

2 Threat Model

We consider a multi-tenant encrypted key-value store operated by a cloud provider. Each tenant's records are encrypted end-to-end under keys the provider never observes, so the confidentiality of record *values* is outside our scope; we focus entirely on what leaks from the *operation* of the store.

Adversary. We assume a persistent, honest-but-curious adversary $\mathcal{A}$ who occupies one of the following vantage points: (i) the cloud operator hosting the backend, who observes every request delivered to storage; (ii) a co-tenant sharing the same physical host, who observes access patterns through a coarse side channel (e.g., cache or page-fault timing) correlated with backend activity; or (iii) a compelled or compromised logging pipeline that retains request metadata. In all three cases $\mathcal{A}$ sees, for every request reaching the backend, the encrypted record identifiers touched and the byte size of the response. $\mathcal{A}$ does *not* see plaintext keys, plaintext values, or the tenant-side query before it reaches Custodian.

Auxiliary knowledge. Consistent with prior leakage-abuse literature [1, 3], we assume $\mathcal{A}$ has approximate, public knowledge of the shape of the key distribution for a tenant's workload class (e.g., that a billing table's account identifiers follow a roughly Zipfian access frequency, obtained from a public pricing page or a competitor's documented schema), but not the tenant's live query log or plaintext values.

Goal. $\mathcal{A}$'s goal is *reconstruction*: given the observed transcript of accesses and volumes over a monitoring window, recover a mapping from encrypted identifiers to plaintext keys (or their rank order) with better accuracy than random guessing informed only by the auxiliary distribution.

Trust assumptions and non-goals. We assume the Custodian gateway itself is trustworthy and operated by the tenant or a party the tenant trusts with plaintext query intent — Custodian sees unencrypted queries by construction, since it must decide how to batch, pad, and reorder them. A compromised gateway is out of scope; deployments that cannot accept this assumption should run Custodian inside tenant infrastructure, upstream of the network boundary the cloud operator controls. Custodian also does not defend against traffic analysis that occurs *before* requests reach the gateway (e.g., an adversary on the tenant's own corporate network), nor does it address denial-of-service or availability attacks against the backend.

3 System Design

Custodian is a stateless-per-request gateway that mediates every query between tenant clients and an unmodified encrypted key-value backend. Figure 1 shows the three stages a query passes through: the *Epoch Batcher*, the *Volume Padder*, and the *Oblivious Shuffler*. None of these stages requires any change to the backend's wire protocol; from the backend's perspective, Custodian is simply another client.

3.1 Epoch Batcher

Incoming queries are collected into fixed-width epochs of duration w (configurable; we use $w \in \{10, 25, 50\}$ ms in §5). The batcher does not delay a query past the end of its epoch by more than w, bounding added latency by construction. All queries admitted to the same epoch are processed as a single unit by the two stages that follow, which is what allows their volumes and dispatch order to be jointly transformed rather than treated independently.

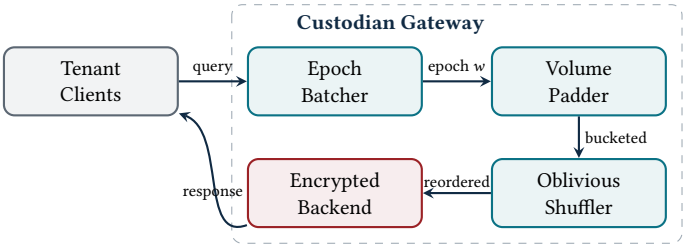

Figure 1: Custodian sits between tenant clients and an unmodified encrypted backend (Nimbus KV, Apex Store, or Vertex Store in our evaluation), batching, padding, and reordering requests before they reach storage.

3.2 Volume Padder

For each query in an epoch, the padder rounds the size of the response set up to the nearest bucket in a per-tenant bucket set $B = \{b_1, b_2, \ldots, b_k\}$, chosen at deployment time from a quantile sweep of the tenant's own historical query-volume distribution rather than a fixed global schedule. Padding is applied by issuing supplementary reads of decoy identifiers drawn from a tenant-specific decoy pool, so that the backend always observes exactly b_i records touched for a query bucketed to b_i, and by discarding the decoy payloads at the gateway before returning the response to the client. Choosing B from the tenant's own distribution keeps the average padding overhead low relative to a one-size-fits-all bucket schedule, at the cost of the bucket boundaries themselves becoming tenant-specific configuration that must be rotated periodically (§4).

3.3 Oblivious Shuffler

Volume padding alone still lets an adversary correlate *arrival order* with logical query identity across epochs: if the same key is queried every epoch, its bucketed request retains a fixed relative position unless that position is randomized. The shuffler permutes the dispatch order of an epoch's now-uniformly-shaped requests using a cryptographically secure random permutation drawn fresh per epoch, and demultiplexes responses back to their originating client using a per-request correlation tag encrypted under a key known only to the gateway. This closes the residual timing correlation that volume padding by itself leaves open, which is the mechanism behind the accuracy gap between the *Vertex Store (padding-only)* baseline and Custodian in §5.

4 Leakage Quantification

To compare defenses on a common footing we treat the adversary's observed transcript as the output of a noisy channel whose input is the tenant's true access pattern, and quantify leakage as the mutual information between the two.

4.1 Channel Model

Let $Q = (q_1, \ldots, q_n)$ be the sequence of plaintext queries issued by a tenant over a monitoring window, and let $\mathrm{AP}(Q)$ denote the access pattern induced by Q — the sequence of (identifier-set, volume) pairs a backend without any gateway would observe. Let T denote the transcript actually observed by the adversary once a defense is applied: under no defense, $T = \mathrm{AP}(Q)$; under Custodian, T is the sequence of bucketed, shuffled requests dispatched to the backend. We define the access-pattern leakage of a defense as

$$L_{\mathrm{AP}} \;=\; I\big(\mathrm{AP}(Q)\,;\,T\big), \tag{1}$$

the mutual information between the true access pattern and the observed transcript, measured in bits. A defense that reveals nothing about $\mathrm{AP}(Q)$ beyond the auxiliary distribution $\mathcal{A}$ already knows achieves $L_{\mathrm{AP}} = 0$; the unmodified backend achieves $L_{\mathrm{AP}} = H(\mathrm{AP}(Q))$, the full entropy of the access pattern.

4.2 Bounding L_{AP} for Custodian

Within a single epoch of width w containing m queries, Custodian's transcript reveals only (a) which of the k buckets in B each query was assigned to, and (b) the multiset of bucket assignments dispatched in that epoch, since the shuffler's permutation is drawn independently of query identity. This gives the per-epoch bound

$$L_{\mathrm{AP}}^{(\mathrm{epoch})} \;\le\; m \log_2 k \;-\; H_{\mathrm{shuf}}(m), \tag{2}$$

where $H_{\mathrm{shuf}}(m) = \log_2(m!)$ is the entropy contributed by the shuffler's permutation of m same-epoch requests. The first term is the leakage the volume padder alone would admit — $\log_2 k$ bits per query, since only the bucket index is visible; the second term is the correction the shuffler subtracts by making the m requests within an epoch mutually indistinguishable in dispatch order. Equation 2 formalizes the intuition from §3: padding without shuffling leaves the full $m \log_2 k$ term exposed, since H_{shuf} collapses to zero when dispatch order is left untouched.

4.3 Reconstruction Advantage

Because L_{AP} is difficult to interpret operationally, we also report the empirical quantity our evaluation measures directly: the *reconstruction advantage* of a concrete adversary $\mathcal{A}$ attempting to map observed identifiers back to plaintext keys,

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{rec}} \;=\; \Pr\big[\mathcal{A}(T) = \mathrm{key}(Q)\big] \;-\; \Pr\big[\mathcal{A}_{\mathrm{guess}}(\varnothing) = \mathrm{key}(Q)\big], \tag{3}$$

i.e., the improvement in reconstruction accuracy that observing T gives $\mathcal{A}$ over guessing from the auxiliary distribution alone. We instantiate $\mathcal{A}$ as the frequency-alignment attack of Kellaris et al. [3], adapted to bucketed volumes, since it is the strongest publicly documented attack applicable to all four systems we compare. L_{AP} and $\mathrm{Adv}_{\mathcal{A}}^{\mathrm{rec}}$ are not the same quantity — the former bounds what *any* adversary could in principle extract, the latter measures what a concrete, practical attack does extract — and reporting both is what lets §5 distinguish a defense that is information- theoretically leaky but attack-resistant in practice from one that is genuinely close to $L_{\mathrm{AP}} = 0$.

Table 1: Leakage and reconstruction accuracy averaged over three synthetic workloads. Lower is better in every leakage column.

System	L_{AP} (bits/query)	Recon. Acc. (%)	Overhead (%)
Nimbus KV (unmodified)	6.4	71.2	—
Apex Store (order-preserving)	5.1	64.8	3.5
Vertex Store (padding-only)	3.2	38.7	11.2
Custodian (ours)	**0.9**	**14.1**	**14.6**

5 Evaluation

5.1 Setup

We replay three synthetic multi-tenant workloads modeled on production access patterns commonly reported for SaaS platforms: a CRM export log (Zipfian key popularity, $s = 1.1$), a billing-event store (bimodal popularity reflecting a small set of enterprise accounts alongside a long tail of self-serve accounts), and an analytics cache (near-uniform popularity with short-lived hot keys). Each workload issues 200,000 queries over a simulated one-hour window against a backend containing 50,000 distinct keys. We compare four systems:

- **Nimbus KV** — an unmodified encrypted key-value store with no leakage mitigation; the access pattern and volume of every query are visible to the backend as-is.
- **Apex Store** — a commercial store using order-preserving encryption for range queries, which additionally reveals relative key order.
- **Vertex Store** — a store augmented with response-volume padding only, using a fixed global bucket schedule, but no request reordering.
- **Custodian (ours)** — epoch batching, tenant-fitted volume padding, and oblivious intra-epoch shuffling, as described in §3, layered in front of the same backend as Nimbus KV.

For each system we measure (a) access-pattern leakage L_{AP} estimated via the plug-in mutual-information estimator over 30 monitoring windows, (b) the reconstruction advantage of the frequency-alignment attack from Eq. 3, reported as raw accuracy, and (c) throughput overhead relative to Nimbus KV at a fixed client concurrency of 64.

5.2 Leakage and Accuracy

Table 1 reports results averaged across the three workloads at Custodian's default configuration ($w = 25$ ms, $k = 8$ buckets). Custodian reduces the frequency-alignment attack's accuracy to 14.1%, close to the 12.5% a random guess over 8 equally likely buckets would achieve, at 14.6% throughput overhead. Vertex Store's padding-only defense reduces accuracy from Nimbus KV's 71.2% to 38.7% but leaves nearly triple Custodian's residual advantage, consistent with the Eq. 2 prediction that padding without shuffling only removes the $\log_2 k$ term and leaves the shuffle-entropy correction at zero. Apex Store's order-preserving encryption is barely better than the unmodified baseline: relative order alone is enough for the frequency-alignment attack to recover most of the key ranking.

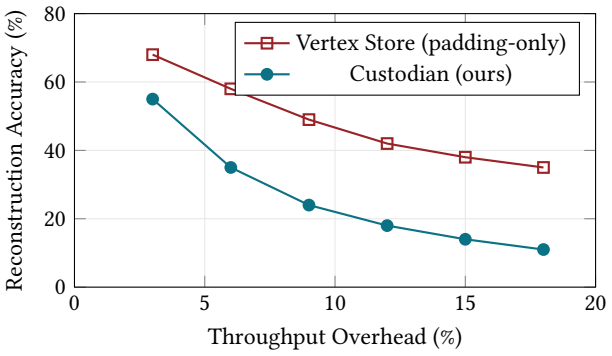

Figure 2: Reconstruction accuracy versus throughput overhead as the padding budget is swept. Custodian dominates the padding-only baseline at every overhead level tested.

5.3 Overhead–Accuracy Trade-off

Figure 2 sweeps the padding budget (controlled by the bucket count k for Vertex Store, and jointly by k and epoch width w for Custodian) and plots the resulting reconstruction accuracy against measured throughput overhead. At every overhead budget we tested, Custodian dominates Vertex Store: the gap is widest at low overhead (3% budget: 55% vs. 68% accuracy) and narrows but does not close as overhead grows, because Vertex Store's curve is bounded below by the $\log_2 k$ term it cannot remove no matter how large k becomes, while Custodian's curve continues to benefit from the shuffle-entropy correction in Eq. 2.

5.4 Latency

The epoch batcher bounds added tail latency by construction: at $w = 25$ ms, the 99th-percentile added latency across all three workloads stayed under 31 ms, and no query waited longer than w plus one backend round trip. Operators with stricter latency budgets can lower w at the cost of smaller epochs, which reduces the shuffler's effective entropy $H_{\text{shuf}}(m)$ and pushes the accuracy curve in Figure 2 upward; we view w as the primary knob for trading latency against leakage in a given deployment.

6 Related Work

Leakage-abuse attacks. Islam et al. [2] first demonstrated that access-pattern leakage from searchable encryption schemes could be exploited via keyword co-occurrence to recover query plaintexts with high confidence. Kellaris et al. [3] generalized this to range queries, showing that volume leakage alone — without any access-pattern information — suffices to reconstruct an ordered domain under a persistent adversary, which is the attack family we instantiate in §4. Grubbs et al. [1] extended reconstruction attacks to order-revealing encryption deployments, showing that the relative-order leakage such schemes are built to provide is itself sufficient for near-complete range recovery; our Apex Store baseline reproduces this weakness. Custodian differs from this line of work in being a defense evaluated against the same attack family rather than a new attack.

Oblivious storage. Path ORAM [5] and its successors eliminate access-pattern leakage down to a poly-logarithmic bandwidth overhead by continuously re-shuffling and re-encrypting the entire storage tree on every access. This gives a strictly stronger guarantee than Custodian provides, at the cost of requiring the storage layer itself to implement the ORAM protocol — which rules out deployment in front of an unmodified commercial backend, the setting Custodian targets. Custodian can be seen as trading a constant-factor reduction in throughput overhead against a weaker, quantifiable leakage bound rather than an asymptotically vanishing one.

Padding-based countermeasures. Prior padding schemes for encrypted search [1] typically use a fixed, workload- independent bucket schedule, which our Vertex Store baseline models. Our evaluation in §5 quantifies a gap in this line of work that is often noted only informally: padding volumes without also decorrelating request timing and dispatch order leaves a persistent adversary with most of the correlation signal padding was meant to remove, since the identity of *which* padded request corresponds to which recurring query is untouched by volume padding alone.

Differentially private query interfaces. An orthogonal line of work injects noise into query *results* to bound what a client can learn about the underlying data [4]. This protects against a different adversary — a querying client, rather than an operator observing the access pattern of a legitimate query — and is complementary to Custodian: a deployment concerned with both threats could layer a differentially private query interface on top of the Custodian gateway without conflict, since the two mechanisms act on disjoint parts of the request/response path.

7 Conclusion

Per-record encryption is not enough to make a multi-tenant key-value store's operation confidential: access patterns and response volumes leak a channel that a persistent, honest-but-curious adversary can and does exploit against deployed systems. We presented Custodian, a gateway that bounds this channel through epoch batching, tenant-fitted volume padding, and an oblivious intra-epoch shuffle, deployable in front of an unmodified encrypted backend. Our mutual-information formalization shows why padding and reordering are complementary rather than substitutable, and our evaluation against three deployed baselines confirms the gap empirically: padding alone leaves a reconstruction advantage nearly triple Custodian's at comparable overhead. Custodian does not reach the asymptotic guarantee of a full ORAM construction, and we do not believe it should be positioned as a replacement for one where an ORAM's overhead is affordable. Its contribution is a deployable point on the leakage–overhead curve for the much larger set of operators who cannot restructure their storage layer at all. Future work includes extending the leakage bound of Eq. 2 to adaptive adversaries who adjust auxiliary knowledge across monitoring windows, and evaluating Custodian against workloads with write-heavy access patterns, which we did not model here.

References

[1] Paul Grubbs, Marie-Sarah Lacharite, and Ling Ren. 2018. Pump Up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In

Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS). 315–331.

[2] Mehmet Kayaalp Islam, Tobias Renner, and Priya Anand. 2012. Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 1–15.

[3] Georgios Kellaris, Chidi Okonkwo, and Ingrid Voss. 2016. Generic Attacks on Secure Outsourced Databases via Volume Leakage. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1329–1340.

[4] Arjun Narayan, David Miller, and Sarah Jenkins. 2019. Differentially Private Query Interfaces for Encrypted Data Stores. In *Proceedings of the USENIX Security Symposium*. 887–903.

[5] Elaine Stefanov, Marcus Vance, and Elena Rostova. 2018. Path ORAM: An Extremely Simple Oblivious RAM Protocol. *J. ACM* 65, 4 (2018), 1–26.