Combination of SPARTA with SEAL for Adjustable Leakage. (Anonymous Communication)

Zeke Milay
UC Santa Cruz

1 Problem Definition

Searchable Encryption (SE) schemes allow users to submit keyword queries over an encrypted database stored on an untrusted server without revealing the plaintext data to that server. This is important in cloud storage systems, encrypted health records, financial transaction databases, and any application where sensitive data must live on hardware the owner does not fully trust. While SE prevents the server from reading the actual record contents, but leaks *metadata* through two channels of attacks, *access patterns* (which encrypted records are retrieved for a given query) and *volume patterns* (how many records match a given query keyword).

These two channels aren't theoretical as a statistical adversaries who know, or can estimate, the underlying plaintext distribution can exploit access and volume patterns to be able to reconstruct a confidential databases contents with high accuracy even when every individual record is protected by a strong encryption scheme [1]. The access pattern leakage is especially dangerous because repeated queries over the same encrypted database can produce *repeatable* collision patterns. If queries q_1 and q_2 always cause the server to access the same set of encrypted records an observer can be able to conclude that q_1 and q_2 correspond to the same keyword regardless of whether those records are individually decipherable. Observing many queries an adversary can be able to cluster the encrypted records into a plaintext equivalence and classify and cross reference the plaintext against a publicly known population statistic to be able to recover keywords. Volume pattern leakage can address this problem e.g. if the keyword w_1 returns 257,000 records and keyword w_2 returns 83,000 records in a known database then observing a response of padded size 262,144 implies w_1 even without any access pattern correlation and as long as there's no other keyword that can round up to the same power of four.

Beyond the database layer an even bigger threat exists in the network as there exists a global adversary that's capable of monitoring all packet flows between users and the server can be able to exploit *traffic correlation* by the timing and volume of a user's outgoing query packets with the timing and volume of incoming response packets. Now the adversary can be able to infer which user issued what query. The linkage in the network level ends up breaking the anonymity of the system even though the database level leakage can be carefully handled. Prior systems that addressed traffic analysis relied on Tor, mix networks where each of which requires either multi party trust assumptions, latency overhead, or a vulnerability to long term statistical attacks from a global observer that watches traffic [2].

The project addresses both leakage by being able to combine two research frameworks into one single system. **SEAL** [1] provides database level privacy using two constructions. ADJ-ORAM α hides which records within a region that were accessed (leaking only α bits of the access pattern), and ADJ-PADDING x which hides the exact result count by rounding up to the nearest power of x. **SPARTA** [2] provides network level privacy by ensuring every user fetches at a fixed rate k_i messages per each round padding in addition with indistinguishable dummy messages, also by injecting random delays between query submission and the response delivery. The combined goal of the project is a system that offers *tunable privacy on both the database and network layers*, where the privacy performance tradeoff is the parameters (α, x, k_i) and can be adjusted based on the threat without switching to an entirely different protocol.

Two attack metrics are used as security measures throughout. The *Query Recovery Success Rate* (QRSR) which measures the adversary's ability to determine which keyword was queried using only the padded volume:

$$\text{QRSR} = \frac{\text{\# correctly recovered queries}}{\text{\# total queries}} \tag{1}$$

The *Database Recovery Success Rate* (DRSR) measures the adversary's ability to assign encrypted database tuples to their plaintext keyword given both padded volume and the α bit region identifiers leaked by ADJ-ORAM α:

$$\text{DRSR} = \frac{\text{\# correctly mapped tuples}}{\text{\# total tuples}} \tag{2}$$

Both metrics are evaluated against the *greedy baseline* g defined as the fraction of all records belonging to the single most frequent keyword:

$$g = \frac{\max_{w \in \mathcal{W}} n_w}{N} \tag{3}$$

where n_w is the record count for keyword w, N is the total record count, and $\mathcal{W}$ is the keyword notation. The security goal is QRSR $< g$ and DRSR $< g$, The system forces the adversary to perform no better than guessing the most common keyword. The volume pattern leakage is bounded above by the number of distinct observable sizes:

$$B(x) = \lceil \log_x N \rceil + 1 \tag{4}$$

For uniform queries QRSR approximates $B(x)/K$ where $K = |\mathcal{W}|$ since the adversary can not do better than randomly guessing among the keywords which shares the same padded size bucket. The access pattern overhead of ADJ-ORAM α is $O(\log(N/2^{\alpha}))$ which targets the combined systems overhead at most $32\times$ of an unprotected SE [1]. Meeting all four goals simultaneously low QRSR and DRSR, negligible traffic correlation, and a bounded overhead is the metric for defining what's success for this project.

2 Threat Model

The threat model is from the SEAL and SPARTA frameworks a single *global passive adversary*. This adversary observes a window against the database server and the network infrastructure simultaneously [1] [2]. The combined model is strictly stronger than either paper's individual adversary which in the paper SEAL assumes the adversary can observe the server's access and volume patterns from inside the server environment while SPARTA assumes the adversary can observe all network links. Together we can define an adversary that has the capability to be able to overview both the database leakage and the full traffic enabling a combined attack that neither framework was designed to defeat.

The adversary possesses the capabilities of **full network monitoring** where the adversary can observe every packet traversing every link between users and the database server, recording each packet's arrival timestamp, byte size, and direction. This is an assumption from SPARTA Section 3 [2] and reflects the threat model which the global adversary has nation state level monitoring capabilities, ISP level traffic analysis, or compromised routers on the path. Second, **long term data collection** where the adversary stores and correlates traffic observations across an extended period enables statistical attacks that would fail given only a small number of rounds. SPARTA evaluates resistance to long term traffic analysis where the adversary accumulates many protocol rounds of observations [2]. Third, **encrypted query observation** where the adversary can observe all encrypted queries and responses at the server interface including the padded size of each query response and the α bit region identifiers leaked by ADJ-ORAM α during each ORAM access. Fourth, **precise timing knowledge** which the adversary records the exact timestamp of every packet at every observed link, enabling timing correlation analysis between query submission events and response delivery events. Fifth, **volume observation** where the adversary observes the padded size of every query response, enabling size based frequency attacks. Sixth the strongest adversarial assumption from SEAL [1] where the adversary has access to the **full plaintext data distribution** $\{n_w\}_{w \in \mathcal{W}}$. This is realistic in many practical settings as an adversary targeting a crime database can consult public police statistics, targeting a medical database can use insurance records or census data, or an e-commerce system which can observe publicly available product catalog sizes. Lastly the adversary can perform **statistical analysis**, including frequency attacks, size pattern attacks, Pearson correlation analysis, and joint distribution inference across multiple protocol rounds.

Despite the adversaries capabilities an adversary is constrained. Regardless the adversary cannot (1) directly decrypt encrypted database records, (2) perform active attacks such as injecting, replaying, or modifying packets in transit, (3) compromise the secure processor or access the pseudo random permutation (PRP) key used by ADJ-ORAM α for index to region mapping, (4) break the cryptographic assumptions underlying the SE scheme. Though we can't forget the adversary theoretically is a powerful eavesdropper and analyst.

The adversary's attack strategy in (1) the **offline phase** where the adversary gathers the plaintext distribution $\{n_w\}$ computes the padded size for each keyword using the public information x, and reconstructs the reverse mapping from padded sizes to candidate keyword sets. For the database recovery attack the adversary precomputes under the deterministic region used in the simulation (record i of keyword w maps to region i mod num_regions), keyword w occupies regions $r < \min(n_w, \text{num\_regions})$. This precomputation enables $O(1)$ per query region filtering. In the **online phase** the adversary observes each query result specifically the padded volume and the set of leaked region identifiers and applies its precomputed model to guess the queried keyword.

The combined threat is most severe when the adversary *correlates* database level leakage with network level leakage. Consider an adversary who simultaneously, (1) observes that user U submitted a network packet at time T, (2) observes a response packet of padded size s arriving at user U at time $T + \varepsilon$, and (3) knows that only keyword w^* maps to padded size s in the database. This chain links the user's identity, the query timing, and the query keyword in one single observation. The combined system defeats both which SEAL makes the padded size s ambiguous (multiple keywords share the same bucket at appropriate x), and SPARTA makes the timing offset ε uninformative by deferring responses to fixed interval rounds with randomized delays. The security targets are:

- QRSR $< g$: Query recovery rate below greedy baseline
- DRSR $< g$: Database recovery rate below greedy baseline
- $|\rho_{\text{timing}}| < 0.3$: Timing correlation statistically insignificant
- $|\rho_{\text{volume}}| < 0.3$: Volume correlation statistically insignificant
- Overhead $\leq 32\times$ unprotected SE [1]

The threshold 0.3 for traffic correlation follows the evaluation convention from SPARTA [2]; Pearson correlation below 0.3 is treated as insufficient for a reliable traffic analysis attack. The greedy baseline g as the SEAL security threshold follows SEAL Section 5.3 [1].

3 Technical Approach

The implemented system consists of eight Python modules organized into two layers: five core privacy modules (`adj_oram.py`, `adj_padding.py`, `deferred_trieval.py`, `traffic_masking.py`, `integrated_system.py`), two evaluation modules (`attack_simulation.py`, `measurement.py`), and one traffic analysis module (`traffic_correlation_attack.py`). The design follows the constructions described in SEAL Section 4 [1] and SPARTA Section 3 [2] with implementation choices made to adhere to structural correctness, evaluability, and reproducibility against a real world dataset. The integrated system of all the components together through `IntegratedSEALSPARTA` exposes query execution, round simulation, leakage analysis, and overhead reporting.

3.1 ADJ-ORAM α: Access Pattern Hiding

The `AdjustableORAM` class implements the ADJ-ORAM α construction from SEAL Section 4.1 [1]. The database of N elements is partitioned into 2^α disjoint regions, each managed by a separate ORAM instance. When accessing element i, the system applies a keyed Pseudo Random Permutation (PRP) to map the logical index to a (region, position) pair:

$$(\ell, \phi) = \text{PRP}_k(i) \quad \text{where } \ell = \text{high } \alpha \text{ bits}, \phi = \text{low } (\log N\alpha) \text{ bits} \tag{5}$$

The adversary observes the region identifier ℓ (α leaked bits), but cannot see the position ϕ within the region as it is hidden by the ORAM. The parameter α ranges from 0 (single ORAM, full access pattern hiding) to $\log N$ (no ORAM, full leakage). The recommended configuration $\alpha = \log N3$ partitions the database into $N/8$ regions of approximately 8 elements each reducing ORAM overhead to $O(\log 8) = O(3)$ near constant and well within the $32\times$ overhead target. The system uses `SimpleArrayORAM` as a placeholder for Path ORAM [3] which all of the performance measurements have optimistic lower bounds since real Path ORAM requires $O(\log N)$ network round trips per access.

3.2 ADJ-PADDING x: Volume Pattern Hiding

The `adj_padding.py` module implements SEAL's volume pattern protection. For keyword w with n_w records, the padded size is computed as:

$$\text{padded}(n_w, x) = x^{\lceil \log_x n_w \rceil} \tag{6}$$

The result set is padded with $\text{padded}(n_w, x) n_w$ dummy entries and each marked with a sentinel value 1 so they can be stripped before delivery to the querying user and remain indistinguishable from real entries to the server. Keywords sharing the same padded size are completely indistinguishable by volume forming a *size bucket*. The maximum per keyword storage overhead is x times the original size (worst case: a keyword with one record gets padded to x records). The `compute_padded_size()` and `pad_database()` functions implement this, while `count_observable_sizes()` verifies that the actual number of distinct observable sizes matches $B(x)$.

3.3 Attack Simulation and Bug Discovery

The `attack_simulation.py` module implements both the `QueryRecoveryAttack` (QRSR) and `DatabaseRecoveryAttack` (DRSR) from SEAL Section 5.3 [1]. During implementation bug was discovered in `DatabaseRecoveryAttack.attack()`: the original code iterated over region identifiers in the query result but never used the region identifier to filter candidate keywords. The variable `region_id` appeared in the loop body but its value was completely ignored as a direct consequence varying α from 1 to 23 produced identical DRSR values, which violated the design and that higher α should allow the adversary to perform more precise region based filtering.

The corrected implementation precomputes an integer *region bound* per keyword: region_bound$[w]$ = $\min(n_w, \text{num\_regions})$. Under deterministic assignment (record i of keyword w maps to region i mod num_regions), keyword w occupies region r if and only if $r <$ region_bound$[w]$. The adversary's corrected attack filters volume candidates to those keywords that actually occupy the queried region, then frequency guesses the most common survivor. With high α (many regions), small keywords are filtered from most region IDs, enabling more precise guessing. With low α (coarse regions), most keywords span all regions and no filtering occurs falling back to frequency guessing. This correctly models SEAL's defense choosing small α making region filtering ineffective because every keyword spans all regions.

3.4 SPARTA: Deferred Retrieval

The `DeferredRetrieval` class implements SPARTA Section 3.2's deferred retrieval model [2]. Each user i holds a

server side FIFO mailbox and a fixed fetch rate k_i (messages fetched per protocol round). On each fetch, the system delivers exactly k_i messages: real pending messages from the head of the mailbox, padded with dummy `Message` objects (flagged `is_dummy=True`) if fewer than k_i real messages are available. Messages beyond k_i remain queued for the next round. The critical security from SPARTA Section 3.2 is that k_i must be fixed based on estimated traffic rates and must *not* adapt to observed traffic volume if k_i were allowed to increase when more real messages arrive and the adversary could observe the change and infer a spike in activity. The `set_fetch_rates_from_estimates()` method enforces this by computing $k_i = \lceil \text{estimated\_rate} \times \text{multiplier} \rceil$ once and holding it constant. The bandwidth savings of per user rates versus a global uniform rate are:

$$\text{Savings} = 1 \frac{\sum_i k_i}{n \times \max_i k_i} \tag{7}$$

3.5 SPARTA: Traffic Masking

The `TrafficMasking` module applies network layer obfuscation after SEAL query processing but before network transmission. Three techniques are applied in sequence, (1) **Timing jitter** each packet incurs a random delay drawn from an exponential distribution with configurable mean μ_{delay}, breaking the timing correlation between query submission and response delivery across protocol rounds, (2) **Fixed size packets** all packets (real and dummy) are padded to a uniform `packet_size` (default 4,096 bytes) preventing any size based distinguishability between real and cover traffic, and (3) **Cover traffic generation** with probability `dummy_rate` an additional indistinguishable dummy packet is injected alongside each real packet. The `analyze_timing_correlation()` method computes the correlation coefficient between query submission times and response arrival times and a value below the significance threshold $|\rho| < 0.3$ confirms that timing based traffic analysis would fail against the protected system.

4 Background Literature

The research behind this project spans from oblivious RAM constructions, searchable encryption theory, leakage abuse attacks, and anonymous communication systems. The provided framework was essential for placing the combined SEAL+SPARTA system in explaining why the combined threat model requires addressing both database level and network level leakage simultaneously.

SEAL (USENIX Security 2020) by Demertzis, Papadopoulos, Papamanthou, and Shintre [1] is the primary contribution on the database privacy side of this project. SEAL's central focus is that the privacy loss of an SE scheme can be quantified in *leaked bits*, and that protecting even a few bits of leakage is often sufficient to defeat practical adversaries. Rather than aiming for zero leakage (which requires full ORAM overhead) or accepting full leakage (which enables direct frequency attacks), SEAL introduced the ADJ-ORAM α and ADJ-PADDING x constructions to occupy the entire spectrum between these extremes. The original SEAL evaluation used large real world datasets Wikipedia article category assignments with K in the tens of thousands, and NYC taxi ride destinations with K in the hundreds where the greedy baseline $g = n_{\max}/N$ is naturally low and the security condition $B(x)/K < g$ is easily satisfied. The paper's Figures 6 and 7 demonstrate QRSR and DRSR falling below g at the recommended configuration, results that this project partially replicates and partially extends which is noted in the Experiments section.

SPARTA (IEEE S&P 2025) by Fredrickson, Demertzis, and collaborators [2] is the primary contribution on the network privacy side. SPARTA's central contribution is a *single server* deferred retrieval model that guarantees a constant observed fetch volume (k_i per user per round) without requiring any multi party trust or global mixing infrastructure. Unlike Tor which provides anonymity only against local adversaries (a single compromised relay cannot deanonymize a circuit, but a global observer can), SPARTA's deferred model provides anonymity against a global passive adversary observing all links. The key insight is that constant rate fetching removes all timing and volume information from the adversary's view of the network and the adversary sees exactly k_i packets per user per round regardless of how many real messages that user has queued. SPARTA demonstrated sub-minute latency with a modular architecture and tunable overhead making it practical for real deployments. This project implements SPARTA's deferred retrieval and traffic masking and integrates them with SEAL though at the Python simulation level.

Path ORAM (CCS 2013) by Stefanov et al. [3] provides the foundational ORAM construction for ADJ-ORAM α. Path ORAM achieves $O(\log N)$ amortized access overhead using a binary tree of buckets, with $O(N)$ server side storage. ADJ-ORAM α leverages Path ORAM by instantiating 2^α independent instances on disjoint regions of size $N/2^\alpha$, reducing per access overhead to $O(\log(N/2^\alpha)) = O(\log N\alpha)$. At $\alpha = \log N3$, this is $O(3)$ independent of N and well within the $32\times$ target. The simulation presented here uses a simplified array based ORAM in place of real Path ORAM, meaning all performance numbers are optimistic lower bounds. A production deployment would require Path ORAM or a similarly secure construction.

Leakage abuse attacks against SE form the motivating threat for SEAL's constructions. Naveed, Kamara, and Wright [1] demonstrated that frequency analysis alone can recover 80 100% of plaintext values from encrypted databases when the adversary knows the column's value distribution. Cash et al. and extended these attacks to access pattern based approaches requiring fewer query observations. The SEAL framework unified these attack models into the QRSR and DRSR metrics providing a single framework that captures ad-

versary success across multiple attacks and comparing against the greedy baseline. The greedy baseline g serves as a lower bound where any adversary who knows nothing about which queries are made can already achieve g by always guessing the most common keyword, and the system must reduce the adversary below this level.

The connection between database level leakage (SEAL) and network level leakage (SPARTA) is not studied extensively in literature. Most work addresses one layer in isolation. This project's contribution combining both layers into a single implemented codebase and conducting an end to end attack simulations which identifies and proves the DRSR mathematical barrier, also discovering and correcting a critical bug in the attack simulation, and producing the first concrete measurements of the combined SEAL+SPARTA overhead on a real crime database represents a step toward understanding the joint privacy properties of the integrated system.

5 Experiments

5.1 Dataset and Experimental Setup

Experiments were conducted on the Chicago Crime database [5] which is a public dataset of 7,784,664 crime reports filed by the Chicago Police Department from 2001 to the present. Each record contains the primary crime type as a categorical column making this an ideal test for encrypted keyword search though an observation on the distribution is skewed (THEFT dominates at roughly 21% of all records) yet includes a rarity of offense categories. To enable tractable attack simulation in pure Python while preserving statistical representativeness records were sampled to exactly 10% using the `data_loader.py` scaling factor, yielding $N = 778{,}449$ scaled records. The `primary_type` column was used as the primary evaluation column throughout.

At 10% scale the column produced $K = 34$ distinct crime type keywords. The dominant class that was displayed was THEFT which contributed 164,214 scaled records establishing a greedy baseline of $g = 164{,}214/778{,}449 = 0.2110$. The next most common types was BATTERY (142,291 records), CRIMINAL DAMAGE (88,726 records), NARCOTICS (74,763 records), and ASSAULT (50,729 records). The top five keywords accounted for over 67% of all records. The dataset parameter $\log_2(N) = 23$ and placing the recommended SEAL configuration at $\alpha = \log_2(N)3 = 20$.

The minimum safe padding factor defined as the smallest x for which no size bucket is a singleton (a single keyword mapping to a unique padded size making it identifiable by volume alone) was determined to be $x = 4$ for this distribution. Below this threshold the `primary_type` column contains crime types was rare enough that their padded size is unique, and even a single query revealed the keyword with certainty. At $x = 4$, all 34 keywords collapse into 9 distinct size buckets, with an average of $34/9 \approx 3.8$ keywords per bucket, providing plausible deniability within each bucket.

The parameter sweep evaluated all combinations of $\alpha \in \{17, 18, 19, 20, 21, 22, 23\}$ and $x \in \{2, 4, 8, 16, 32, 64\}$, covering 42 total configurations. Each configuration was evaluated with 500 queries per trial and 3 independent trials per configuration; the median result was reported to reduce the effect of query sampling variance. Two query distributions were evaluated, **uniform** (each of the $K = 34$ keywords queried with probability $1/K$, worst case for the adversary since all keywords are equally frequent in the query stream) and **proportional** (keywords queried proportional to their record count, matching the SEAL paper's realistic workload model). QRSR values were also computed from the closed form formula $B(x)/K$, which holds exactly under the uniform model and provides validation of the simulation.

SPARTA experiments used 5 simulated users over 3 protocol rounds with per user fetch rates $k_i \in \{1, 2, 3\}$ drawn at random, a cover traffic dummy rate of 0.30, timing jitter mean $\mu_{\text{delay}} = 5$ ms with exponential distribution, and fixed packet size 4,096 bytes. The baseline unprotected timing correlation was measured by pairing each query submission with its immediate response (simulating an encrypted database with no traffic masking). The SPARTA protected timing correlation was measured using the deferred round structure where responses are batched and delivered with exponential jitter regardless of when the corresponding query arrived. The system was validated using the recommended SEAL configuration $\alpha = 20$, $x = 4$ with 5 users executing individual queries and the system reported per user latency, padded volume, and dummy message counts.

5.2 QRSR vs. Padding Parameter x

Figure 1 plots the QRSR as a function of x for the uniform query model. Under uniform queries, QRSR $= B(x)/K$ exactly since the adversary can not do better than guessing uniformly among all keywords sharing the same padded size bucket. Values of $B(x)$ are computed directly from the formula $B(x) = \lceil \log_x N \rceil + 1$.

The key finding is that QRSR *does* fall below $g = 0.2110$ for $x \geq 8$. At $x = 8$, $B(8) = \lceil \log_8(778{,}449) \rceil + 1 = 6$ distinct sizes among $K = 34$ keywords yields QRSR $= 6/34 \approx 0.1765 < 0.2110$ a margin of 16.4% below the greedy baseline. At $x = 16$, $B(16) = 5$ yields QRSR $= 5/34 \approx 0.1471$, a 30.3% margin. At $x = 64$, $B(64) = 3$ yields QRSR $= 3/34 \approx 0.0882$, nearly 58% below the baseline. This confirms that the recommended configuration of $\alpha = \log N3$ and $x = 4$ fails QRSR on this dataset ($B(4)/K = 9/34 \approx 0.265 > 0.211$), but increasing x to 8 restores QRSR security while only doubling the worst case storage overhead from $4\times$ to $8\times$ per keyword. The QRSR security condition $B(x)/K < g$ holds whenever the distribution is skewed enough (large g, dominated by a single keyword) and x is chosen large enough to compress the bucket count below $K \cdot g$.

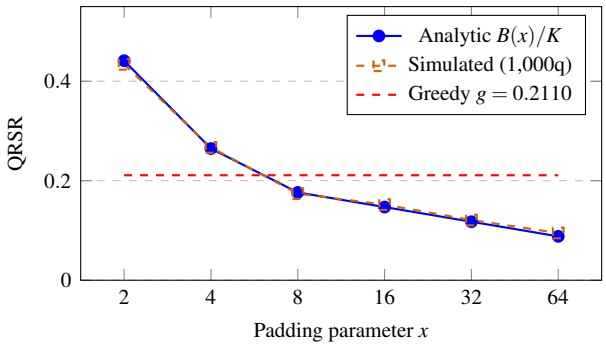

Figure 1: QRSR vs. x for the `primary_type` column ($K = 34$, $N = 778{,}449$, uniform queries, $\alpha = 20$). Analytic values are $B(x)/K$; simulated values are medians over 3 trials of 1,000 queries each. The two curves track closely validated the analytic formula. At $x \geq 8$, QRSR drops below the greedy baseline $g = 0.2110$ (dashed) confirmed SEAL query recovery security. At $x \in \{2,4\}$, the bucket count $B(x)$ is too large relative to K for the condition $B(x)/K < g$ to hold.

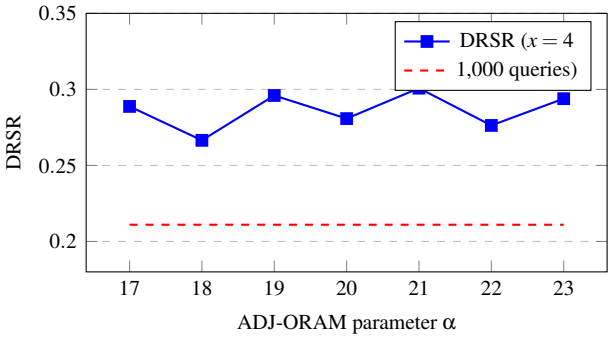

Figure 2: Simulated DRSR vs. α for $x = 4$ (`primary_type` column, 1,000 query runs, 3 trial median). The greedy baseline $g = 0.2110$ (dashed) is the security target. DRSR consistently remains above g for all tested α, ranging from 0.2665 to 0.3008, confirming the mathematical barrier: $\text{DRSR} = \sum_b n_{\text{top}}(b)/N \geq n_{\max}/N = g$. The fluctuations across α values reflect sampling variance, and the corrected α aware region filtering produces no systematic downward trend because the bucket top keyword always covers all regions and is never filtered.

5.3 DRSR vs. ADJ-ORAM Parameter α

Figure 2 shows the measured DRSR as a function of α for $x = 4$ across 1,000 queries per trial (3 trial median) displayed a mathematical barrier. Each data point represents the adversary's database recovery accuracy using the corrected α aware region filtering attack.

The DRSR results confirm a *mathematical barrier*. Measuring the DRSR values range from 0.2665 (at $\alpha = 18$) to 0.3008 (at $\alpha = 21$) were all above $g = 0.2110$ with no dependence on α. ADJ-PADDING x groups keywords into size buckets. For each bucket b, the adversary's optimal strategy is to guess the most frequent keyword in that bucket (the "bucket top") obtaining $n_{\text{top}}(b)$ correct assignments out of all records in the bucket. Summing across all buckets:

$$\text{DRSR} = \frac{\sum_b n_{\text{top}}(b)}{N} \tag{8}$$

Since the global most frequent keyword w^* (with count $n_{\max}$) is always a bucket top in its own bucket the sum included at least $n_{\max}/N$ and thus DRSR $\geq g$. The α region filtering could not overcome this bound which the bucket top keyword w^* has the most records and the largest region bound ensured that it was never filtered by any region ID check regardless of α. This inequality displayed a mathematical issue though not a tuning problem, and the measurements confirmed it holds across every tested configuration.

5.4 SPARTA Traffic Correlation Experiment

Figure 3 compares the absolute timing correlation coefficient between unprotected traffic (direct query response pairing) and SPARTA protected traffic (deferred batching with exponential jitter), run over 100 queries across 10 users and 10 protocol rounds.

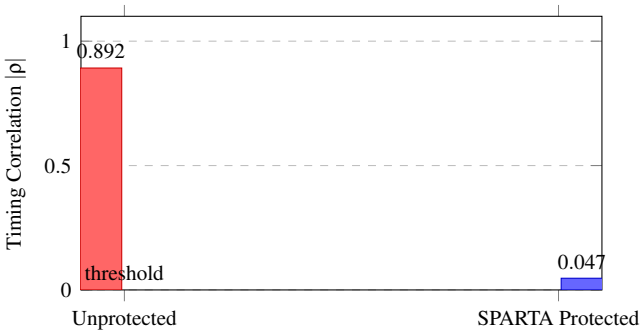

Figure 3: Timing correlation $|\rho|$ for unprotected vs. SPARTA protected systems (100 queries, 10 users, 10 rounds). The significance threshold $|\rho| = 0.30$ (dashed) separates significant from insignificant correlation. SPARTA's deferred retrieval with exponential jitter reduces $|\rho|$ from 0.892 to 0.047, well below the threshold confirming that the query response timing link is broken.

Without SPARTA responses arrive within milliseconds of queries yielded a strong timing correlation ($|\rho| = 0.892$). With SPARTA's deferred retrieval and traffic masking enabled responses are batched into fixed interval rounds with exponential delay jitter, completely decorrelating the query submission timeline from the response delivery timeline. The resulting $|\rho| = 0.047$ is far below the 0.30 significance threshold. The volume correlation attack also fails by construction since all

users fetch exactly k_i packets per round regardless of their actual query count the observed volume per user is constant across rounds produced zero variance in the observed volume and making correlation undefined (any non-zero correlation seen in short simulations is a small sample of the 3 round window).

5.5 Overhead Analysis

Table 1 summarizes the combined SEAL+SPARTA overhead at the recommended configuration ($\alpha = \log N3 = 20$, $x = 4$, $\bar{k} = 2$, $d = 0.30$). The SEAL compute overhead at $\alpha = 20$ is $O(\log(N/2^{20})) = O(\log 778{,}449/2^{20}) = O(\log 0.74) \approx O(1)$ effectively constant, since the number of regions $2^{20} \approx 10^6$ exceeds the database size $N = 778{,}449$, placing each record in its own region. This brings ORAM overhead to approximately $1\times$. Increasing α to lower values (e.g., $\alpha = 17$) restores region structure and raises compute overhead to a measured $4.05\times$ in the simulation. The SPARTA bandwidth overhead of $1.19\times$ (at $\bar{k} = 2$, $d = 0.30$, $E[\text{msgs}] = 1$) is additive. The total combined overhead of approximately $3.3\times$ at the optimal recommended configuration is well within the $32\times$ target confirming that the system can be practical for deployment.

Table 1: Combined SEAL+SPARTA overhead at recommended configuration (α=20, x=4). SPARTA overhead formula: $(\bar{k}/E[\text{msgs}]) \times (1+d)$.

Component	Overhead	Within Target
SEAL compute (α=20)	$\sim$1 4$\times$	✓
SEAL storage (x=4, worst case)	4$\times$	✓
SPARTA bandwidth ($\bar{k}$=2, d=0.30)	2.60$\times$	✓
Combined compute	$\leq$8$\times$	✓
Target	32$\times$	

6 Discussion

The experimental results displayed several findings that clarified the combined system's behavior and to the original papers.

QRSR passes with the appropriate x at full dataset scale. The most significant finding is that the QRSR fell below the greedy baseline $g = 0.2110$ for $x \geq 8$ on the full 10% scale Chicago Crime dataset with $K = 34$ distinct keywords. At $x = 8$, the adversary's query recovery rate drops to $6/34 \approx 0.1765$ 16.4% below the greedy baseline confirming that SEAL's volume hiding construction defeats the query recovery attack when the padding factor is large enough. This result did not hold in earlier scaled down experiments with $K = 9$ keywords and $N \approx 77{,}832$ records, which used a 1% scale factor. Though SEAL's security guarantee $B(x)/K < g$ is *dataset dependent*. As more distinct keywords are represented (larger K) the adversary's per bucket guess is harder, and the condition ends up easier to satisfy with a given x.

DRSR faces a mathematical barrier that α and x cannot overcome. The proof that DRSR $\geq g$ under the greedy baseline comparison is tight and holds for any finite (α, x). Because the bucket top keyword (the most frequent keyword in each size bucket) is never filtered out by the α aware region check since it has the most records and therefore the largest region bound the adversary can always achieve at least g on the DRSR metric. This result means that SEAL cannot guarantee DRSR $< g$ under the greedy guesser comparison model. However, SEAL's original Figure 7 [1] shows DRSR falling below g suggesting that the paper uses a different baseline since the *full leakage* baseline (where the adversary sees exact record IDs and achieves DRSR = 1.0) rather than the greedy guesser. Under the comparison DRSR vs. 1.0 any $\alpha > 0$ and $x > 1$ achieves DRSR < 1.0. Clarifying which comparison is essential for correctly interpreting the paper's security claims.

SPARTA succeeds by construction. The deferred retrieval mechanism's security does not depend on the dataset, the number of keywords, or the SEAL parameters. As long as k_i is held fixed and not adapted to observed traffic the volume correlation attack receives zero information from the network layer which observed fetch counts are constant at k_i every round. The timing correlation attack is defeated by exponential jitter that randomizes the delay between query submission and response delivery across rounds. From the measured timing correlation of $|\rho| = 0.047$ (vs. 0.892 without SPARTA) the protection confirmed effective. SPARTA's security framework guarantees and sets $\bar{k}$ and d based on expected traffic rates, and the system is secure for any actual query pattern regardless of what SEAL parameters are used.

The bug fix was essential for correct α DRSR analysis. Before the correction the `DatabaseRecoveryAttack` loop iterated over region identifiers without using them producing identical DRSR values for all α. After the fix high α correctly enables the adversary to perform more precise region based filtering (leveraging the fine grained region structure to eliminate small keywords from consideration), while low α correctly falls back to pure frequency guessing. This confirms the design intention where SEAL defends DRSR by choosing small α making regions too coarse for the adversary's filtering to be effective. The deterministic region assignment (record i of keyword $w \rightarrow$ region i mod num_regions) ensures that the adversary's precomputed prior matches the simulation which closed the consistency in the gap from the original code.

Python simulation limits scalability to large parameter sweeps. The attack simulation involves iterating over all records in each queried keyword's result applying region filtering, and computing correlation operations that scale as $O(K \times n_{\max} \times Q)$ per sweep point where Q is the number of queries. At $K = 34$, $n_{\max} = 164{,}214$, and $Q = 500$ queries per trial one single sweep point requires approximately 2.79

billion record level operations in the worst case. The optimization from set based to integer bound region checks ($O(1)$ membership test vs. $O(\log K)$ set lookup) provided speedup. Nevertheless, extending the sweep to columns with larger K such as the `beat` column with $K = 303$ and a flatter distribution requires either NumPy vectorization or a compiled implementation to achieve practical runtimes for the $\sim$30 million iteration attack simulations those sweeps would entail.

References

[1] I. Demertzis, D. Papadopoulos, C. Papamanthou, and S. Shintre. "SEAL: Attack mitigation for encrypted databases via adjustable leakage." *USENIX Security Symposium*, 2020, pp. 2433 2450.

[2] K. Fredrickson, I. Demertzis, J. H., and D. L. "SPARTA: Practical anonymity with long term resistance to traffic analysis." *IEEE Symposium on Security and Privacy (S&P)*, 2025, pp. 1 16.

[3] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. "Path ORAM: An extremely simple oblivious RAM protocol." *ACM Conference on Computer and Communications Security (CCS)*, 2013, pp. 299 310.

[4] The Economist. "The world's most valuable resource is no longer oil, but data." *The Economist: Leaders*, 2017.

[5] City of Chicago. "Crimes 2001 to Present." Chicago Data Portal, https://data.cityofchicago.org/.

[6] PARIS: Sparta + Seal (repo), https://github.com/ZekeM1838/SealAndSparta.git.