Organizing Records for Retrieval in Multi-Dimensional Range Searchable Encryption

Mahdieh Heidaripour[1] [a], Ladan Kian[1] [b], Maryam Rezapour[2] [c], Mark Holcomb[1] [d],
Benjamin Fuller[2] [e], Gagan Agrawal[3] [f], and Hoda Maleki[1] [g]

[1]Augusta University. fmheidaripour,lkian,mholcomb, hmalekig@augusta.edu
[2] University of Connecticut. fmaryam.rezapour, benjamin.fullerg@uconn.edu
[3]University of Georgia. gagrawal@uga.edu

Abstract: Storage of sensitive multi-dimensional arrays must be secure and efficient in storage and processing time. Searchable encryption allows one to trade between security and efficiency. Searchable encryption design focuses on building indexes, overlooking the crucial aspect of record retrieval. Gui et al. (PoPETS 2023) showed that understanding the security and efficiency of record retrieval is critical to understand the overall system. A common technique for improving security is partitioning data tuples into parts. When a tuple is requested, the entire relevant part is retrieved, hiding the tuple of interest. This work assesses tuple partitioning strategies in the dense data setting, considering parts that are random, 1-dimensional, and multi-dimensional. We consider synthetic datasets of $2, 3$ and 4 dimensions, with sizes extending up to 2M tuples. We compare security and efficiency across a variety of record retrieval methods. Our findings are:

1. For most configurations, multi-dimensional partitioning yields better efficiency and less leakage.

2. 1-dimensional partitioning outperforms multi-dimensional partitioning when the first (indexed) dimension is any size as long as the query is large in all other dimensions.

3. The leakage of 1-dimensional partitioning is reduced the most when using a bucketed ORAM (Demertiz et al., USENIX Security 2020).

1 Introduction

Scientific data is often organized as multi-dimensional arrays (Rusu, 2023). Datasets' size necessitates efficient solutions regarding storage, processing time, and communication. Searchable encryption (Song et al., 2000; Boneh and Waters, 2007; Bellare et al., 2008; Chase and Kamara, 2010; Tu et al., 2013; Bøsch et al., 2014; Fuller et al., 2017; Kamara et al., 2022) allows a client C to outsource a database DB, to a server S. The client should then be able to retrieve records corresponding to a query q efficiently from the DB without the server learning about the contents of DB or q.

[a] https://orcid.org/0009-0007-1836-4717
[b] https://orcid.org/0009-0008-2202-0762
[c] https://orcid.org/0009-0009-6172-7557
[d] https://orcid.org/0009-0006-1830-2578
[e] https://orcid.org/0000-0001-6450-0088
[f] https://orcid.org/0000-0002-2609-1428
[g] https://orcid.org/0000-0002-9486-1164
    M. Heidaripour and L. Kian co-corresponding authors.

This work considers multi-dimensional range queries (Markatou and Tamassia, 2019; Falzon et al., 2020; Akshima et al., 2020; Markatou et al., 2021; Markatou et al., 2023). For a number of dimensions ℓ, a database $\mathsf{DB} = (\mathsf{DB}_1, ..., \mathsf{DB}_n)$ is a collection of tuples DB_j where $\mathsf{DB}_j \in \prod_{i=1}^{\ell} [1, m_i] \times \mathsf{R}$: where m_i is the maximum value for dimension i. We use the following terms:

1. The values x_i are the dimensions of a tuple,

2. The value m_i is the domain of a dimension, and

3. R is associated data and is called a record.

A range query $q := \prod_{i=1}^{\ell} [a_i, b_i]$ finds all $\mathsf{DB}_q := \{\mathsf{DB}_j | 1 \le i \le \ell, a_i \le x_{j,i} \le b_i\}$. Searchable encryp-tion design usually focuses on building an index (and corresponding protocol) that calculates the subset of records that have to be retrieved. That is, they design an index retrieval mechanism called RetrieveIndexes that returns a set $I \subseteq [1, n]$ of matching records. A second, often unspecified mechanism is used to retrieve the actual records. We call this method RetrieveData. In an full system, these protocols are used in sequence

for each query.

Recent work (Gui et al., 2023a; Gui et al., 2023b) shows the efficiency and security aspects of record retrieval are crucial to understanding the overall system. Given the unequal attention paid to the two stages, this work focuses exclusively on RetrieveData assuming a correct RetrieveIndexes stage. Like search, initialization consists of two components, SetupIndexes, and SetupData.

1.1 Prior Work on Record Retrieval

We provide a brief overview of methods used to provide security during record retrieval. We then discuss the prior combination of these techniques. Techniques for hiding record access are shuffling, caching, partitioning, and query flattening (Grubbs et al., 2020; Maiyya et al., 2023).
We consider the following research question:

> How are leakage and efficiency impacted by the organization of tuples into parts?

We make five simplifying assumptions:

1. During retrieve index, the system retrieves the correct records; this excludes systems that use approximate range covers (Demertzis et al., 2016; Falzon et al., 2023)

2. During retrieve data, no "fake queries" are issued. This excludes systems that flatten the distribution (Grubbs et al., 2020; Maiyya et al., 2023).

3. That each tuple appears in a single part.

4. Empty tuples are only used when the dataset size is not divisible by part size.

5. That the partition is static; only parts are moved.

Assumption 1 is made for scoping reasons, however, considering approximately correct systems such as those that use range covers is an important piece of future work discussed in Section 8. Assuming a query distribution without flattening (Assumption 2) is necessary to study the research question. Assumptions 3-5 are satisfied by the cryptographic retrieval methods discussed below. We introduce representative record retrieval mechanisms. These mechanisms are used to assess the leakage of a partition strategy. These approaches are: static shuffling, oblivious, n-bucketed oblivious, and SWiSSSE (Gui et al., 2023b). We consider three partitions: row-based shuffling or DRW (shown in Fig. 1), slab-based shuffling or SLW (shown in Fig. 2), and record-wise shuffling or RCW (shown in Fig. 3).
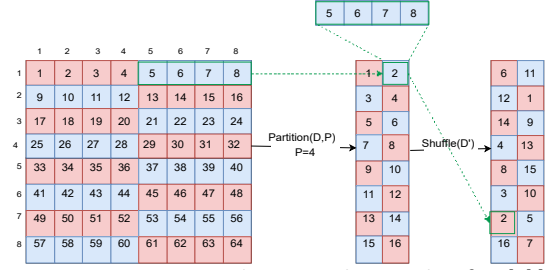

Figure 1: DRW- a two-dimensional example of a [8] [8] dataset. The numbers inside cells show the logical address location of the tuples in the storage. For example, tuples (5,1),(6,1),(7,1),(8,1), with addresses 5,6,7 and 8 in the original dataset, go to location 13 in the shuffled dataset.
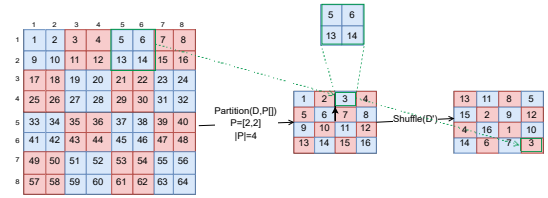
Figure 2: SLW- a two-dimensional example of a [8] [8] dataset. The numbers inside cells show the logical address location of the tuples on the storage. For example, tuples (5,1),(6,1),(5,2),(6,2), with addresses 5,6,13, and 14 in the original dataset, go to location 16 in the shuffled dataset on the right side of the figure.

1.2 Our Contribution

We evaluate the cross product of the above data retrieval mechanisms and partitioning strategies. Our evaluation is with respect to efficiency and security on dense, synthetic two-, three-, and four-dimensional data of size up to 2M records. In Section 3, we argue for metrics to assess the security when one instantiates record retrieval with each candidate system (shown in Table 1). For Shuffled and SWiSSSE we adopt the average number of parts, for the two ORAM systems we adopt the entropy of number of parts. Our evaluation supports three major conclusions.

For Shuffled and SWiSSSE, the page organization that requests the fewest number of pages also has the best security. However, efficiency and security are not aligned when using (n-bucketed) ORAM. For both ORAM variants, one can actually reduce the entropy of response size by having most queries request the maximum number of parts. RCW, which requests nearly all parts, has the lowest values for both entropy metrics despite its poor efficiency.

We consider four query types (Section 5):

1. Isotropic: each dimension has the same width.

2. Bisected anisotropic: separate dimensions into two parts with equal widths for each.
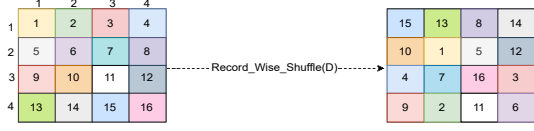
Figure 3: RCW: A two-dimensional example of a [4][4] dataset. The numbers inside cells show the logical address location of the tuples on the storage. We shuffle the locations of the tuples. For example, tuple (3,2) with address 7 in the original dataset goes to location 10 in the shuffled dataset on the right side of the figure. Parts are then created based on new organizations of equal size. (Any partition strategy has the same outcome after a random permutation.)

Retrieval Mech.	Leakage	
	Profile	Metric
Shuffled	Access Pattern	Parts/Query
SWiSSSE	SWiSSSE	Parts/Query
ORAM	Volume	H(Parts)
bucket ORAM	Bucket Volume	H(Parts)

Table 1: Association between storage mechanism and assessed leakage metric. For all schemes, we consider the average # parts returned as the efficiency metric. For the n-bucketed ORAM, the volume is padded to the next power of n. H is the entropy function on the distribution of part numbers.

3. Gradual anisotropic: dimension width gradually reduces.

4. Outlier anisotropic: all but one dimension are of equal width. In these queries, we always make the first dimension differ. We split the query type by whether the first dimension is smaller or larger than the other dimensions, called min and max, respectively.

Our experiments demonstrate that SLW partitioning is generally the best with respect to efficiency and security across various query shapes, with one exception. DRW partitioning exhibits higher performance and lower leakage on outlier queries where dimensions $2; :::; \ell$ have large sizes (and the size of the first dimension is arbitrary).

Both DRW and SLW outperform RCW as expected and as shown in Table 2. Table 2 shows the percentage of tuples retrieved from the parts that were in the specified query. RCW usually requires at least three times as many parts as the other two methods. Our detailed results, presented in Tables 3, show that in general **SLW** presents superior security and efficiency compared to **DRW**. There are two notable exceptions:

1. DRW is sensitive to the width of dimensions $i > 1$, performing well when the query is large in other dimensions. (When the width of each dimension is the same in all dimensions, which dimension is indexed is irrelevant.) This is displayed in the relevant query % in outlier queries in Table 2. In

Dim.		Relevant Tuple %		
	Query Type	RCW	DRW	SLW
4	Isotropic	15.6	43.6	41.2
	Bisected	12.3	35.7	40.6
	Gradual	4.0	16.1	23.9
	Outlier Min	5.8	76.5	20.2
	Outlier Max	.1	.4	2.8
3	Isotropic	26.3	66.5	73.3
	Bisected	2.2	6.3	8.1
	Gradual	1.2	4.4	6.7
	Outlier Min	.5	11.5	3.1
	Outlier Max	.0	.2	1.0
2	Isotropic	34.0	66.3	85.1
	Bisected	25.1	49.7	80.4
	Gradual	24.5	49.2	80.0
	Outlier Min	.5	64.4	8.0
	Outlier Max	.5	.5	8.1

Table 2: Relevant tuple percentage > 1M record datasets across number of dimensions. Size of desired record set over size of returned record set. Summarizes Tables 3. Bolded entries are at least 10% better than other methods for the same data and query set. Dimension 3/2 datasets are detailed in the online version of this work (Heidaripour et al., 2024)

Outlier Min, where dimension 1 is small compared to other dimensions, DRW performs much better than SLW. However, when dimension 1 is larger than other dimensions SLW performs better than DRW though all methods perform poorly.

2. DRW receives more security benefit from the use of n-bucketed volume than SLW. This is due to more variety in part numbers for DRW.

The above schemes are implemented, code is published in a public GitHub repository.

2 Preliminaries

Let $l \in N$ be a security parameter. Throughout all algorithms are collections of algorithms indexed by security parameter l. However, l is often omitted from notation for simplicity.

For integers $a; b$ let $[a; b] = \{x \in Z \mid a \le x \le b\}$ and let $[b]$ be shorthand for $[1; b]$. We consider a database DB where each tuple DB_i consists of ℓ dimensions and an associated record. For $1 \le i \le \ell$, let m_i denote the maximum value of the ith dimension which ranges from $[m_i]$. A database $DB = (DB_1; :::; DB_n)$ is an ordered collection of tuples where each $DB_j \in \tilde{O}^\ell [m_i] \times R$: where $1 \le x_i \le m_i$. For a query $\vec{q} = (a_1; b_1); :::; (a_\ell; b_\ell)$ where $1 \le a_i \le b_i \le m_i$ let

$$DB_{\vec{q}} = \{DB_j \mid 8 1 \le i \le \ell; a_i \le x_{j;i} \le b_i\}:$$

Separating lookup and retrieval   Let DB be a database of size N. We separately define two functions:

1. $Index : [m_1] \cdots [m_\ell] \to 2^{\lfloor \log N \rfloor}$.

2. $Storage : 2^{\lfloor \log N \rfloor} \to (R \times \bot)^N$.

We provide formal definitions of searchable encryption in the full version of this work (Heidaripour et al., 2024). Our definition considers four operations: 1) SetupIndexes generating an encrypted index set, 2) SetupData created an encrypted array of records, 3) RetrieveIndexes which finds the indices and 4) RetrieveData which retrieves the tuples.

3   Prior Work: Retrieval Mechanisms and Leakage Metrics

This section provides an overview of retrieval methods discussed in the Introduction (shuffled, ORAM, n-bucketed ORAM, SWiSSSE). This cross-section of retrieval methods covers the main techniques of caching, shuffling, and partitioning. (As a reminder, we ignore query flattening, as it destroys optimization of partitioning based on query load.) Throughout our discussion, we consider a persistent adversary (Grubbs et al., 2017) as codified in the full version of this work (Heidaripour et al., 2024). Roughly, the adversary 1) sees the encrypted database and all communication from the client, 2) follows the protocol, and 3) does not specify the query distribution.

Before introducing defensive schemes, we introduce common terminology used to describe leakage functions.

- Access Pattern. Reveals the (persistent) identifiers of all returned parts.

- Search/Equality Pattern. If queries are equal.

- Co-occurrence Pattern. Reveals parts that are jointly returned by queries.

- Volume Pattern. Reveals the number of parts returned by each query.

3.1   Prior Retrieval Mechanisms and Leakage Analysis

We consider four record retrieval methods. Each of these mechanisms yields a different leakage profile. In Section 3.2, we introduce 1-dimensional metrics to assess leakage.

Shuffled   During SetupData one selects a random permutation of parts that is consistent throughout setup and search. This method has access pattern leakage.

ORAM   ORAM (Goldreich, 1987; Goldreich and Ostrovsky, 1996) systems for RetrieveData leaks how many records are accessed (assuming the tuple and parts are fixed size).

n-bucketed Oblivious SEAL (Demertzis et al., 2020) allows fine-tuning of leakage during the setup phase. The construction of SEAL involves adjustable ORAM (a) and adjustable padding (n). They use two techniques: memory partitioning and padding the number of accesses from an ORAM. For n, one retrieves $n^{\lceil \log_n k \rceil}$ records instead of n. We focus on the ORAM padding technique because we already consider memory partitioning for all systems.

SWiSSSE (Gui et al., 2023b) SWiSSSE can be seen as a partial ORAM where the access pattern is not completely oblivious, the main benefit is that it only requires two rounds.

3.2   Prior Attacks and Leakage Metrics

We argue for metrics to assess a partitioning strategy across retrieval methods. This analysis considers both the leakage function and existing attacks. Leaker (Kamara et al., 2022) provides an overview of attacks that perform data and query-recovery (without the use of either auxiliary or known data and queries).

Existing access pattern attacks rely on building a co-occurence matrix and the success of the attack depends on the number of items returned together. In SWiSSSE, only a single row of a co-occurrence matrix is exposed to the adversary. Gui et al. evaluated this leakage, and their attack recovered frequently returned items. We adopt the average number of accessed parts as our metric for Shuffled and SWiSSSE.

Current (bucketed) ORAM attacks (Grubbs et al., 2018) are for a single dimension. They rely on observing all possible ranges. When one moves from one-dimensional to multi-dimensional ranges the number of ranges increases from N^2 to $N^{2'}$ making it more difficult for the adversary to observe each range. We adopt the entropy of the number of returned parts as our evaluation metric for ORAM and n-bucketed ORAM.

4   Partition Organization

We implement a multi-dimensional search to measure different partitions using a three-phase multimap as follows:

1. Creation of an Index Structure: In this initial phase, an index structure is established by running SetupData, and SetupIndexes. We use a k-d tree (Bentley, 1975) for finding the appropriate

records, all exact methods are equivalent for evaluating the record retrieval stage.

2. Data Shuffling: The second phase entails shuffling the source data on disk.

3. Dictionary Generation: In the third phase, a dictionary is created where a range from the index structure is linked with a set of values derived from the newly shuffled data.

We consider three shuffling schemes.

Divided-Row Shuffling (DRW) In Divided-Row Shuffling we adopt a row-based storage order for the dataset. It is shown in Figure 1. DRW sequentially divides the dataset into partitions of size $|P|$ through the stored source data. For example, consider a dataset with dimensions $[8][8]$ and a partition size of $|P| = 4$. In this scenario, we obtain 16 partitions that transform the dataset into a $[8][2]$ matrix. Subsequently, the shuffling is performed by applying a pseudoran-dom permutation to the parts.

The shuffled matrix forms the foundation for the subsequent steps. From this point, we can construct a multi-dimensional indexing structure by utilizing the starting points of each partition. In the illustrative example provided in Figure 1, the starting points for the partitions are indicated as $(1; 1)$, $(5; 1)$, $(1; 2)$, $(5; 2)$, and so forth, extending up to $(1; 8)$ and $(5; 8)$.

Slab-Wise Shuffling (SLW) In Slab-Wise Shuffling (SLW) we adopt a partitioning strategy for datasets that are Slab-oriented, denoted as P, and are characterized by a d-dimensional partition shape of $[p_1]$ $[p_2] \cdots [p_d]$. This partitioning is depicted in Fig-ure 2. In the specific illustration presented, a dataset initially sized $[8][8]$ is segmented into a collection of 16 distinct $[4][4]$ matrices. This results in a trans-formation to a $[4][4]$ dataset structure. Following this Slab-oriented organization, we begin a shuffling process targeting this newly arranged dataset. The shuffling is guided by the locations of the individual slabs within the structure.

Record-Wise Shuffling (RCW) RCW is a record-wise permutation of the source data. Record-Wise Shuffling unlike the preceding methods, requires no logical grouping of values; hence the record-wise des-ignation.

Fig. 3 illustrates a two-dimensional example. In this example, we have a $[4][4]$ two-dimensional ar-ray of tuples. We store the tuples in row-based storage and assign logical location addresses to the tuples in a row-based order from 1 to 16. Then we permute the locations, mapping them so they are stored in a

new spot. Afterward, we build the multi-dimensional index structure based on the points in their new loca-tions.

5 Query Distribution

We consider four types of queries based on the rel-ative queried width of each dimension. Isotropic queries, denoted as Q_{iso}, have a uniform scaled length in each dimension. For a query $q = [a_1; b_1] \cdots [a_d; b_d]$ let $\forall i; w_i := |a_i - b_i|$. A query q belongs to the set Q_{iso} if $q \in Q_{iso} \iff \forall i; j \; |w_i - w_j| \leq e$ where e is a deviation parameter. In our implementation, we set $e = 0$, as we only consider queries that are per-fectly isotropic. Anisotropics have scaled lengths of their intervals that vary across dimensions. The set of anisotropic queries is the complement of isotropic queries: $Q_{aniso} = Q_{iso}^{\complement}$. We categorize anisotropic queries into Bisected, Gradual, and Outlier.

- Bisected anisotropic queries (BAQ) partitions the dimensions into two parts each with a width gap of at most e within each part. In four dimensional datasets, this means two sets of two dimensions have the same width.

- Gradual anisotropic (GAQ) queries scale the width of each dimension down by a factor of c. In our implementation, we randomly choose $c \in \{1; 2; \cdots; \frac{DB.max}{n}\}\frac{1}{w}$ where w is the width of the query's widest dimension.

- Outlier anisotropic queries (OAQ) are isotropic in all but one dimension which has either a smaller or larger width. We call a OAQ query min if the dimension of differing size is smaller and max if it is larger. In our implementation, the outlier is always dimension 1 to highlight the differences between our shuffling techniques.

6 Evaluation Methodology

We experimentally evaluate the performance of our schemes. We used a randomly generated dataset be-cause the actual values in the datasets are unlikely to impact our measurements. Our conclusions depend on the dataset's size, number of dimensions, and the width of each dimension. We consider six datasets in the full version of this work (Heidaripour et al., 2024). We focus on a four-dimensional dataset of sizes $1048576 = 32^4$ with a DRW row of size 4096, and SLW slabs of size 8^4. We issue 10K queries. To the best of our knowledge, no standard benchmarks

of queries on array-based data are available. A uniform distribution was used for our experiments to generate Hyper-Rectangular range query samples for each query shape according to the shapes discussed in Section 5. For dense data, only the relative size of each dimension matters; we believe our query shapes explore this parameter space well.

Experimental Setup We implemented our schemes in Python 3.10.12 and conducted all our experiments on a computer with 8-core processors and 16G RAM. We utilized Python's cryptography library version 40.0.2 (pyca/cryptography, 2023) and employed AES-128 encryption in XTS mode with a 256-bit key size for symmetric encryption, using line positions as tweaks. Our code is published in a public GitHub repository.

7 Experiment Results

Our primary efficiency measure is the number of parts accessed per query. For best security, one seeks a small number of returned parts with small variance. This saves both on memory access for the return while presenting less leakage. We computed the leakage metrics introduced in Section 3, shown in Table 3. Figure 4(a) illustrates the distribution of part access numbers for the [32][4] dataset with respect to the four different query shapes: isotropic queries, bisected anisotropic queries, gradual anisotropic queries, and outlier anisotropic queries using violin graphs. The width of a violin graph represents frequency while the y-axis represents the number of parts accessed in a query. In all of our results, we present the size of the query for comparison (Figure 4(b)). A solution with 100% relevant tuple percentage would have these two identical graphs.

Table 3 shows the (n-bucketed) entropy and average number of parts accessed across all queries. For n-parts, we have rounded the number of parts to the closest power of n = 2, which at least the number of accessed parts, and then computed our leakage metrics over them. As discussed in Section 3, Access Pattern and SWiSSSE leakage are minimized by minimizing the number of partitions that are accessed through the retrieval process. To minimize (bucketed) volume leakage, one desires low entropy in the volume of accessed parts.

The use of n-parts is most useful when there is a small increase in the amount of required parts but a large decrease in the entropy of parts returned. As an example, the results in Table 3 shows that for DRW when we have outlier anisotropic queries max, the average number of parts increases by 10% while entropy

Query	Shuf.	Parts			n-Parts		
		Avg	s	H	Avg	s	H
Isotropic	RCW	230	71	1.7	230	66	.65
Avg = 150K	DRW	82	68	5.5	110	95	2.8
STD = 207K	SLW	87	92	3.5	105	100	2.9
Bisected	RCW	250	38	1.5	250	32	.28
Avg = 120K	DRW	85	69	6.7	110	91	2.6
STD = 180K	SLW	74	74	4.2	99	95	2.7
Gradual	RCW	250	33	.76	250	30	.13
Avg = 40.9K	DRW	62	48	5	83	65	2.4
STD = 36K	SLW	42	30	3.6	60	43	2.5
Outlier Min	RCW	260	0	0	260	0	0
Avg = 61K	DRW	19	8.8	2.6	21.8	10	1.5
STD = 28K	SLW	74	23	.6	74	23	.6
Outlier Max	RCW	171	91	4.9	200	97	.8
Avg = 730	DRW	40	14	2.9	43	15	.94
STD = 710	SLW	6.3	4.3	1.3	6.3	4.3	1.3

Table 3: Leakage Metrics for 4D datasets. n set to 2, other small values displayed similar trends. H is the entropy of the (bucketed) part distribution.

of parts decreases by roughly 66%. On the other hand, for SLW for isotropic queries, the average number of parts increases by 22% with the entropy decreasing by only 17%.

As shown in Table 3, across query shapes RCW has lowest value of n-bucketed entropy, followed by DRW, and then SLW.

It can also be inferred from the table that the benefit of n-bucketing for the DRW is more significant than for the SLW as the entropy decreases more in DRW. Looking at Figure 4(a), this can be the result of having a more continuous distribution in DRW than in the SLW. For outlier anisotropic queries we see the most discontinuity in SLW distribution (4(a)-(IV) and (V)), and this is when bucketing does not have any effect on the entropy of accessed parts.

DRW and SLW have very different performance and security on outlier queries, they differ by a factor of 2 on both average and entropy of parts. For isotropic queries, where the query has the same width along all dimensions on 4D data, DRW performs better than SLW. On the security side, DRW has a lower average number of parts but higher entropy.

However, this improvement in average returns and entropy found from the different shuffling schemes is not consistent with the other query shapes or other datasets (Heidaripour et al., 2024); For all other query types, the average number of parts and entropy for SLW is smaller compared to DRW.

8 Conclusion

Prior study of secure multi-dimensional storage focuses on indexing structure. This work focuses on
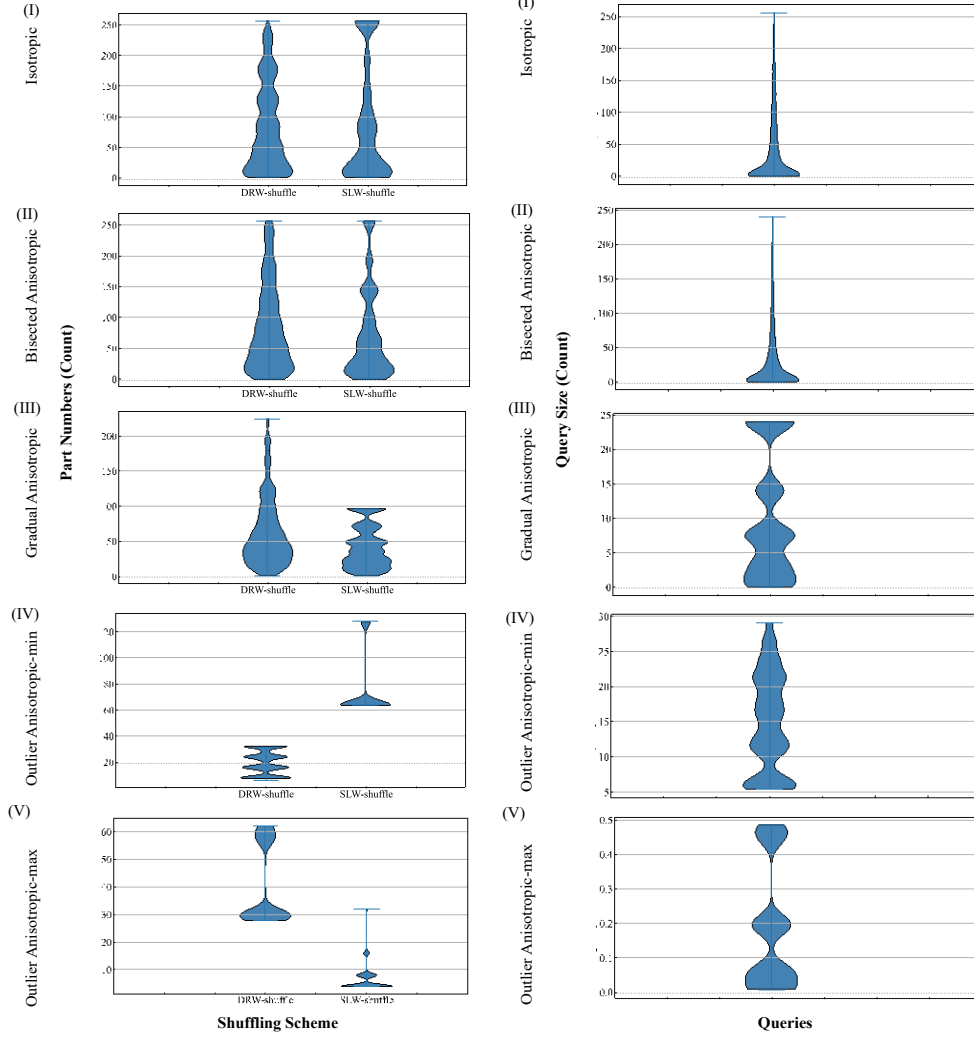
(a) Part numbers distribution for the 4D-[32]^4 dataset.

(b) Query Size Distribution.

Figure 4: (a)-Part numbers count per query shape for the DRW and SLW schemes, [32]^4 4D dataset. The x-axis represents the dataset of part numbers returned as query results per shuffling method (b)-Query size distribution. The x-axis represents the queries dataset

the less-charted territory of optimizing storage and retrieval steps. Our primary study is the differences in efficiency and security of data retrieval for searchable encryption mechanisms across datasets and query shapes. In most scenarios, reducing the average and variance of returned parts improves both efficiency and security of the system. That is, efficiency and security are aligned. DRW demonstrates superior performance only in the setting where dimensions i > 1 are large in each query. This is counter intuitive, we usually organize data based on the most important dimension, here the width of the non-indexed dimensions are critical.

We recommend that future research investigates the interactions between different shuffling strategies and index structures that allow for false positives, such as the single range cover (Falzon et al., 2022). Such systems usually reduce the number of possible ranges that are queryable. A natural solution is to organize data according to these queryable ranges. However, unlike the organizations considered in this work, tuples usually are in more than one range cover.

7

REFERENCES

Akshima, Cash, D., Falzon, F., Rivkin, A., and Stern, J. (2020). Multidimensional database reconstruction from range query access patterns. Cryptology ePrint Archive.

Bellare, M., Fischlin, M., O'Neill, A., and Ristenpart, T. (2008). Deterministic encryption: Definitional equivalences and constructions without random oracles. In CRYPTO, pages 360–378, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9):509–517.

Boneh, D. and Waters, B. (2007). Conjunctive, subset, and range queries on encrypted data. In Theory of Cryptography Conference, pages 535–554. Springer.

Bosch, C., Hartel, P., Jonker, W., and Peter, A. (2014). A survey of provably secure searchable encryption. ACM Computing Surveys (CSUR), 47(2):1–51.

Chase, M. and Kamara, S. (2010). Structured encryption and controlled disclosure. In Asiacrypt, pages 577–594. Springer.

Demertzis, I., Papadopoulos, D., Papamanthou, C., and Shintre, S. (2020). SEAL: Attack mitigation for encrypted databases via adjustable leakage. In USENIX Security, pages 2433–2450.

Demertzis, I., Papadopoulos, S., Papapetrou, O., Deligiannakis, A., and Garofalakis, M. (2016). Practical private range search revisited. In ACM SIGMOD/PODS Conference.

Falzon, F., Markatou, E. A., Cash, D., Rivkin, A., Stern, J., and Tamassia, R. (2020). Full database reconstruction in two dimensions. In CCS, pages 443–460.

Falzon, F., Markatou, E. A., Espiritu, Z., and Tamassia, R. (2022). Range search over encrypted multi-attribute data. Proc. VLDB Endow., 16(4):587–600.

Falzon, F., Markatou, E. A., Espiritu, Z., and Tamassia, R. (2023). Range search over encrypted multi-attribute data. In VLDB. https://eprint.iacr.org/2022/1076.

Fuller, B., Varia, M., Yerukhimovich, A., Shen, E., Hamlin, A., Gadepally, V., Shay, R., Mitchell, J. D., and Cunningham, R. K. (2017). SoK: Cryptographically protected database search. In IEEE Security and Privacy, pages 172–191.

Goldreich, O. (1987). Towards a theory of software protection and simulation by oblivious rams. In STOC, pages 182–194.

Goldreich, O. and Ostrovsky, R. (1996). Software protection and simulation on oblivious rams. Journal of the ACM (JACM), 43(3):431–473.

Grubbs, P., Khandelwal, A., Lacharité, M.-S., Brown, L., Li, L., Agarwal, R., and Ristenpart, T. (2020). Pancake: Frequency smoothing for encrypted data stores. In USENIX Security, pages 2451–2468.

Grubbs, P., Lacharité, M.-S., Minaud, B., and Paterson, K. G. (2018). Pump up the volume: Practical database reconstruction from volume leakage on range queries. In CCS, pages 315–331.

Grubbs, P., Ristenpart, T., and Shmatikov, V. (2017). Why your encrypted database is not secure. In Proceedings of the 16th workshop on hot topics in operating systems, pages 162–168.

Gui, Z., Paterson, K. G., and Patranabis, S. (2023a). Rethinking searchable symmetric encryption. In IEEE Security and Privacy.

Gui, Z., Paterson, K. G., Patranabis, S., and Warinschi, B. (2023b). SWiSSSE: System-wide security for searchable symmetric encryption. PoPETS.

Heidaripour, M., Kian, L., Rezapour, M., Holcomb, M., Fuller, B., Agrawal, G., and Maleki, H. (2024). Organizing records for retrieval in multi-dimensional range searchable encryption. Cryptology ePrint Archive, Paper 2024/635. https://eprint.iacr.org/2024/635.

Kamara, S., Kati, A., Moataz, T., Schneider, T., Treiber, A., and Yonli, M. (2022). Sok: Cryptanalysis of encrypted search with LEAKER - a framework for LEakage AttacK Evaluation on Real-world data. In Euro S&P.

Maiyya, S., Vemula, S. C., Agrawal, D., El Abbadi, A., and Kerschbaum, F. (2023). Waffle: An online oblivious datastore for protecting data access patterns. Proceedings of the ACM on Management of Data, 1(4):1–25.

Markatou, E. A., Falzon, F., Espiritu, Z., and Tamassia, R. (2023). Attacks on encrypted response-hiding range search schemes in multiple dimensions. PoPETS.

Markatou, E. A., Falzon, F., Tamassia, R., and Schor, W. (2021). Reconstructing with less: Leakage abuse attacks in two dimensions. In CCS, pages 2243–2261.

Markatou, E. A. and Tamassia, R. (2019). Full database reconstruction with access and search pattern leakage. In International Conference on Information Security, pages 25–43. Springer.

pyca/cryptography (2023). Python cryptography library 40.0.2. https://cryptography.io.

Rusu, F. (2023). Multidimensional array data management. Foundations and Trends® in Databases, 12(2-3):69–220.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In IEEE Security and Privacy, pages 44–55. IEEE.

Tu, S. L., Kaashoek, M. F., Madden, S. R., and Zeldovich, N. (2013). Processing analytical queries over encrypted data. In VLDB. Association for Computing Machinery (ACM).