TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Efficient Multi-dimensional Range Searchable Encryption for Large Databases

Rafail-Athanasios Demertzis

Thesis Committee:

Professor Minos Garofalakis (ECE)
Associate Professor Antonios Deligiannakis (ECE)
Associate Professor Vasilis Samoladas (ECE)

September 2019

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αποδοτική Κρυπτογραφημένη Αναζήτηση για Πολυδιάστατες Ερωτήσεις Εύρους για Μεγάλες Βάσεις Δεδομένων

Δεμερτζής Ραφαήλ-Αθανάσιος

Εξεταστική Επιτροπή:

Καθ. Μίνως Γαροφαλάκης (ΗΜΜΥ)

Αναπλ. Καθ. Αντώνιος Δεληγιαννάκης (ΗΜΜΥ)

Αναπλ. Καθ. Βασίλειος Σαμολαδάς (ΗΜΜΥ)

September 2019

Abstract

In Searchable Encryption (SE), a data owner outsources an encrypted set of documents to a server, with the purpose of enabling the server to answer keyword queries in a private manner. Recently, SE has been extended to support more expressive private queries, such as range, aggregate, boolean, phrase, substring, wildcard queries and a wide range of SQL queries.

In this work, we introduce the notion of Multi-Dimensional Range SE (MRSE) schemes and we provide novel constructions (MRSE-A, MRSE-B, MRSE-C) by extending/modifying Logarithmic-SRC-i_1 and SRC-i_2 RSE schemes, proposed by Demertzis et al.(SIGMOD 2016, TODS 2018) to multiple dimensions. We demonstrate that, given any secure SE/RSE scheme, the challenge boils down to (i) formulating leakages that arise from the index structure, (ii) improving the index space, and (iii) minimizing false positives incurred by the space reduction. We further propose a new MRSE scheme utilizing a powerful cryptographic tool, Oblivious Random Access Memory (ORAM), and introduce the notion of Oblivious-MRSE (OMRSE) that reduces the leakages introduced by MRSE schemes. We demonstrate a surprising finding; we experimentally show that our strongly secure OMRSE scheme requires significantly less space than the proposed MRSE schemes, and has comparable/better search performance (up to 2× slow-down,up to 100× speed-up) with MRSE schemes when are tuned to have similar space demands with the OMRSE scheme.

Περίληψη

Στο σχήμα της κρυπτογραφημένης αναζήτησης (SE), ο client διαθέτει την κρυπτογραφημένη μορφή ενός σετ δεδομένων την οποία αποστέλλει στο server ώστε να υποστηριχθεί ένα πρωτόκολλο ερωτημάτων το οποίο θα επιτρέπει την ιδιωτικότητα και την προστασία των δεδομένων. Πλέον, μετά από μια σειρά πρόσφατων επεκτάσεων, το σχήμα της κρυπτογραφημένης αναζήτησης υποστηρίζει ποικίλες μορφές ερωτημάτων, όπως τα range, aggregate, boolean, phrase, substring, wildcard queries καθώς και πολλά από τα SQL ερωτήματα που συναντάμε στα παραδοσιακά συστήματα βάσεων δεδομένων.

Σε αυτή την εργασία, επεκτείνουμε την ιδέα της κρυπτογραφημένης αναζήτησης σε πολυδιάστατα ερωτήματα. Πιο συγκεκριμένα, επικεντρωνόμαστε σε πολυδιάστατα ερωτήματα εύρους και προτείνουμε τρία νέα σχήματα (MRSE-A, MRSE-B, MRSE-C) τα οποία προκύπτουν μετά από κατάλληλες επεκτάσεις των Logarithmic-SRC-i_1 και SRC-i_2 RSE σχημάτων που προτάθηκαν στη δουλειά των Demertzis et al.(SIGMOD 2016, TODS 2018) για μονοδιάστατα ερωτήματα εύρους. Οι βασικές δυσκολίες στη σχεδίαση ενός ασφαλούς SE/RSE σχήματος είναι: (α) η ακριβής διατύπωση για τη μορφή της πληροφορίας που μπορεί να υποκλέψει κάποιος πιθανός κακόβουλος χρήστης παρατηρώντας τη δομή του index, (β) η μείωση της χωρητικότητας των indexes περιορίζοντας όμως ταυτόχρονα των αριθμό των false positives η αύξηση των οποίων προκύπτει ως συνέπεια αυτής της μείωσης. Στη συνέχεια προτείνουμε ένα νέο MRSE σχήμα, το Oblivious-MRSE (OMRSE) που βασίζεται σε ένα ισχυρό κρυπτογραφικό πρωτόκολλο, το Oblivious Random Access Memory (ORAM) και σκοπεύει στη μείωση της πληροφορίας που διαρρέεται από τα MRSE σχήματα. Πειραματικά δείχνουμε ότι το OMRSE, που δίνει πιο ισχυρά privacy guarantees, χρειάζεται λιγότερη χωρητικότητα από τα MRSE ενώ όταν τα συγκρίνουμε υπό τις ίδιες προδιαγραφές μνήμης το OMRSE έχει συγκρίσιμη ή/και καλύτερη απόδοση (έως και $2\times$ slow-down, έως και $100\times$ speed-up).

Acknowledgements

Foremost, I would like to express my sincere gratitude to my thesis advisor Prof. Minos Garofalakis for his continuous guidance and support throughout the fulfillment of this thesis. I would also like to thank Prof. Antonios Deligiannakis and Prof. Vasilis Samoladas for accepting to be in my committee.

A big thanks goes to my parents, Michalis and Chrysanthi, as well as my older brother, Ioannis for their unconditional love and support.

Last but not least, I am thankful to all of my friends in Chania, Athina, Eirini, Eleftheria, Giannis, Iro, Marina, Periklis and Thanos for all the great experiences we shared together during our studies and for standing by me.

Contents

List of Figures

Chapter 1

Introduction

Data breaches occur with an alarming frequency. According to estimates, 5.5 million records are stolen on a daily basis. European Union takes data protection and people's privacy very seriously enforced the General Data Protection Regulation (GDPR) on May 25th 2018. GDPR gives data protection authorities more robust powers to tackle non-compliance, including significant administrative fining capabilities of up to 20 million Euro (or 4% of total annual global turnover, whichever is greater) for the most serious infringements. Thus, there is a pressing need for companies to keep data encrypted on premise and, more importantly, on third-party cloud platforms. Currently, companies use ad hoc solutions that are based on a hot and cold setting, in which they store users' data encrypted in disks (cold storage) and they fetch, decrypt and keep those data unencrypted in-memory (hot storage) in order to perform the necessary computations. The above heuristic solutions are both vulnerable to a plethora of security threats and lack efficiency due to the encryption/decryption overhead.

Towards improving the efficiency of the above naive approaches have been proposed new works for private search and privacy preserving database management systems. These solutions reach the desired performance at the cost of lacking rigorous security guarantees, e.g. CryptDB [1] and Monomi [2] support a wide-range of SQL queries using deterministic (DET) and order preserving encryption (OPE)[1]. A recent work of Naveed et al.[3] shows that these systems are not reliable, due to their vulnerability to severe attacks, which allow an attacker to take advantage of the leaked statistical and order information and **decrypt the actual encrypted records**.

An alternative is to use more sophisticated cryptographic solutions, such as *Searchable Encryption (SE)* schemes, in order to reach better security guarantees. SE, proposed by Song et al.[4] in

[1]Deterministic encryption leaks the distribution of the input data. Order preserving encryption leaks both the distribution of the input data and the order of the data.

2000, enables a data owner to outsource a document collection to a server in a private manner, so that the latter can still answer keyword search queries. In a typical SE scheme, the data owner prepares an encrypted index which is sent to the server. To perform a keyword search, given a keyword w, a token $t(w)$ is sent by the data owner to the server that allows the server to retrieve pointers to the encrypted documents containing the keyword w, while leaking some information, e.g., the search (whether a search query is repeated) and the access (encrypted document that satisfy the query) or volume patterns (the size of the query result). Recently, SE was extended to support more expressive queries, such as boolean, range, substring, wildcard, phrase queries and SQL queries [5–9]. SE is used not only as a secure alternative to CryptDB-based solution, but also as an efficient alternative to very expensive approaches such as oblivious RAMs [10, 11] and fully-homomorphic encryption [12, 13]. In fact, SE schemes have been proven to be very practical at the expense of well-defined leakages.

Despite the recent advancements in SE, there are certain query types that they have not meet yet realistic security-efficiency goals required from real-world applications. For instance, it is still an open problem to efficiently and securely support privacy preserving *Multi-dimensional Range (MDR) queries*. In theory, the work of Kamara and Tarik [9] providing a wide-range of SQL queries can also provide a solution for MDR queries; however the proposed solution for MDR cannot be considered as a viable solution since it requires to first linear scan each individual dimension and then produce the final result computing the intersection. Additionally, MDR queries can be supported combining the recent works of [7, 8, 14] that support private singe-dimensional range queries with the works of [5, 6] that support private conjunctive queries. Again, the latter approach cannot be deployed in practice, since the works of [5, 6] in order to support conjuctive queries with minimum leakage and efficient search time require exponential (to the number of dimensions) size indexes.

A very important question is why we do not combine SE with index structures, such as quad trees, k-d trees, R-Trees, grid files, etc., which have been proposed in the literature for plaintext MDR queries, especially in the area of spatial databases. Integrating these index structures with SE achieves in practice scalable setup costs, but traversing the index becomes the main challenge. That is because traversing the index requires multiple rounds of interaction between the client and the server, which not only introduces inefficiencies, but most importantly increases the leakage since each traversal of the index depends on the distribution of the dataset . This means that for each traversal not only we leak information about the search/access patterns, but we leak how exactly the search algorithm traverses the tree from the root node to the matched leaf nodes for each given query, i.e., it leaks the identifiers of all the nodes in the paths traversed by the search algorithm for each given query.

Scheme	Storage	Search Time	False Posit	Interactions	Leakage/Security
MRSE-A	$O(dm + dnlogn)$	$O(min(r_i))$	$O(min(r_i))$	2	2
MRSE-B	$O(dm + nlog^d n)$	$O(min(r_i))$	$O(min(r_i))$	2	1
MRSE-C	$O(nlog^d m)$	$O(R + r)$	$O(R + r)$	d	3
OMRSE	$O(n)$	$O(n)$	$O(n)$	$O(r \log n)$	0

n: dataset size, r : result size, m: the sum of the domain sizes of the d dimensions, i.e, $m = \sum_{\forall i \in [1,d]} m_i$

R: the maximum query range size of the individuals d dimensions.

TABLE 1.1: Summary of Our MRSE Schemes

In the literature, there is a plethora of works that address the private MDR problem proposing solutions without rigorous security analysis, such as [15–17]. We do not consider the latter works as reference points to our work, since the lack of formal security analysis renders them both incomparable and it is unknown if they can be used in practice[2].

In this work. We propose the first MRSE schemes extending existing private singe-dimensional RSE schemes to multiple dimensions [8, 14] (Logarithmic-SRC, SRC-i$_1$,SRC-i$_2$). We also provide the first Oblivious MRSE (OMRSE), which (i) reduces the leakages of the MRSE schemes, (ii) minimizes the setup costs, and (iii) achieves at the same time practical search efficiency. Our constructions with their performance and security characteristics are summarized in Table 1.1 and discussed in detail in Section 3. In particular:

1. Our first scheme MRSE-A creates d instances of the Logarithmic-SRC-i$_2$ scheme [8, 14]; one for each dimension. Given a d-dimensional range query the client breaks the query to d sub-queries one for each dimension. Then, she executes a multi-dimensional range query in two rounds. In particular, in the first round she interacts with the d instances of the Logarithmic-SRC-i$_2$ scheme in order to retrieve the size of the result for each dimension. In the second round, she fetches the result from the Logarithmic-SRC-i$_2$ instance/dimension with the minimum result size. The final answer contains false positives which are filtered-out by the client. The search complexity of MRSE-A is proportional to the smallest result size of the d sub-queries, i.e., $O(min(r_i))$ for $i \in [1, d]$.

2. Our MRSE-B scheme improves the security of MRSE-A, and experimentally reduces the number of false positives (see Section 4) by modifying the Logarithmic-SRC-i$_2$ scheme of MRSE-A—which is based on the Logarithmic-SRC scheme. The underlying data structure of the aforementioned scheme is based on a modification of a range tree [20]. The modified range tree assumes a full binary tree, where each internal node contains the union of its

[2]We illustrate the importance of rigorous security analysis of newly proposed schemes by the following example: The paper of Karras et al.[18] was proposed in SIGMOD 2016 without formal provable security guarantees; the work of Horst et al.[19] in SIGMOD 2017 provided successful attacks against the schemes proposed by Karras.

descendants. Note that in Logarithmic-SRC-i$_2$ version for the one-dimensional case the required space to store this data structure is $O(m + n \log n)$. We can trivially extend the above data structure to d dimensions in a similar manner as we extend the range tree to multiple dimensions. MRSE-B scheme maintains one additional index for each dimension which will be used in order to store/search the individual result sizes for each dimension —these indexes will be used in a similar manner as the first-round of MRSE-A.

3. Our MRSE-C scheme is the first scheme that can bound the number of false positives to be proportional to actual result and range size at the expense of more leakage and increased number of interactions. MRSE-C scheme based on modifying Logarithmic-SRC-i$_1$ scheme to multiple dimensions, in similar manner as in MRSE-B, i.e., by extending the range-tree like structure of Logarithmic-SRC-i$_1$ to multiple dimensions.

4. The main drawback of the aforementioned approaches is the increased size of the encrypted indexes. For instance, for MRSE-C we observe that the size of the index is $O(n \log^d m)$, where d denotes the number of supported dimensions and m the maximum domain size of all dimensions, and therefore this cost clearly becomes a bottleneck. For instance, in the most secure/efficient scheme of [8], i.e. Logarithmic-SRC-i$_1$, and for $d = 1$ (one-dimensional range queries), the size of the encrypted indexes for 5 million tuples is 14 GB. Extending the Logarithmic-SRC-i$_1$ in MRSE-C we can observe that the size of the encrypted indexes for $d = 2$ using the same schemes will be at least 300 GB, and similarly for $d = 3$ it is estimated to reach 6 petabytes. We tackle this problem introducing a new trade-off between the space and false positives in MRSE-A, MRSE-B, and MRSE-C. Instead of storing all the levels of the used range-tree like structure, we store only s evenly distributed levels for each dimension; the missing levels increase the number of false positives.

5. Our OMRSE scheme is based on the use of Oblivious RAM schemes, ideas from Oblivious Data Structures [21], and R-tree indexes. In particular, we use the aforementioned building blocks, and we create an oblivious R-tree, which can be used for supporting strongly secure MDR queries. Our OMRSE leaks only the size of the returned result, i.e., it does not leak the access/search pattern leakages as the MRSE schemes. Additionally, the index size is $O(n)$, and the search performance is depended on the R-tree implementation and used ORAM—which introduces a polylog n multiplicative overhead (using PathORAM [11] and setting carefully the block size this overhead can be $O(\log n)$). We experimentally show that our strongly secure OMRSE scheme requires significantly less space than the proposed MRSE schemes, and has comparable/better search performance (up to 2× slow-down,up to 100× speed-up) with the MRSE schemes—when the MRSE schemes are tuned to have

similar space demands (e.g., keeping only 2 evenly distributed levels per dimension) with the OMRSE scheme.

Chapter 2

Background

2.1 Preliminaries

2.1.1 Searchable Encryption (SE) Scheme

SE definitions. Let D be a collection of *documents*, where a document can be any data item, even a tuple. Each document $d \in D$ has a unique id, which is an *alias* that allows easy mapping to d. Every d is also associated with a unique identifier $d.id$ and a set of keywords from a dictionary Δ, each denoted as $d.w$. We represent by $id(w)$ the ids of the documents that contain w and $|id(w)|$ the number of documents that contain w. We also define $n \triangleq \sum_{w \in \Delta} |id(w)|$ as the size of dataset D, i.e., the number of all $(d.id, d.w)$ pairs for all $d \in D$. SE schemes focus on building an *encrypted index* I on the document ids. For simplicity, we concentrate only on the ids, since the actual documents are encrypted independently and stored at the server separately from I; once some id is retrieved during search, the server can send the corresponding document to the owner, who decrypts in a final step that is *orthogonal* to the SE instantiation.

An SE *protocol* involves an *owner* and a *server* and consists of the following algorithms:

$k \leftarrow \mathsf{Setup}(1^\lambda)$ A probabilistic algorithm run by the owner before commencing the system. It takes as input security parameter λ and outputs a secret key k.

$I \leftarrow \mathsf{BuildIndex}(k, D)$ A probabilistic algorithm run by the owner prior to sending its data to the server. It takes as input the secret key k and the data collection D, and outputs an encrypted index I built on the document ids. Index I is sent to the server, along with the actual encrypted documents.

$t \leftarrow \mathsf{Trpdr}(k, w)$ A deterministic algorithm executed by the owner when issuing a query. It takes as input key k and keyword w, and outputs a token t.

$X \leftarrow \mathsf{Search}(t, I)$ A deterministic algorithm run by the server to retrieve the ids of the documents containing the query keyword. It takes as input a token t corresponding to the query keyword and the encrypted index I, and outputs a set X of document ids.

In state-of-the-art SE constructions [6, 22–27], I is essentially an encrypted *inverted index*, which allows efficient retrieval of the document id list corresponding to the query keyword. The token t constitutes auxiliary information that allows the server to *partially decrypt* only the index components that lead to the retrieval of the result ids. However, once these index portions are decrypted, they become permanently known to the server. In other words, SE inherently introduces certain information *leakage*.

An *ad-hoc* way of defining security would be to outline a set of adversarial attacks, and prove that the scheme is robust against these attacks. This is dangerous as we cannot anticipate the types of attacks an adversary is able to launch. A *rigorous* way to define security is to *formulate* the leakage, and *prove* that the adversary learns nothing more than this leakage. Curtmola et al. [22] introduced a framework for achieving this, following the seminal *ideal-real paradigm* by Goldreich [28]. In particular, after formulating leakage, we define two *games*. The *real* is essentially the execution of the actual SE protocol. The *ideal* is a *simulation* of the real, i.e., an attempt to "fake" the real game, knowing only the formulated leakage. Finally, we prove that an adversary can *distinguish* the output of the first from that of the second with only *negligible* probability. Intuitively, this means that the adversary indeed does not learn anything more than the leakage, otherwise he would be able to distinguish the real from the ideal execution with non-negligible probability.

We focus on *semi-honest*, *adaptive* adversaries. "Semi-honest" means that the adversary is curious to infer information during the execution of the protocol, but does not deviate from the protocol. "Adaptive" means that the adversary attempts to learn information even in between query executions, and may adaptively select the next query based on the previous ones. A non-adaptive adversary submits all queries before starting to learn information. Clearly, adaptive adversaries are more realistic in database applications where the queries are not presented all at once to a system.

For completeness, in Figure 2.1 we present the SE ideal-real games for (semi-honest) adaptive adversaries, as introduced in [29]. In $\mathbf{Real}_{\mathsf{SE},\mathcal{A}}$, an adversary $\mathcal{A}$ interacts with the actual SE protocol, *choosing* the initial document set and (adaptively) the keyword queries. The adversary

$\mathbf{Real}_{\mathsf{SE},\mathcal{A}}(k)$	$\mathbf{Ideal}_{\mathsf{SE},\mathcal{A},\mathcal{S}}(k)$
$k \leftarrow \mathsf{Setup}(1^\lambda)$	
$(D, st_\mathcal{A}) \leftarrow \mathcal{A}_0(1^\lambda)$	$(D, st_\mathcal{A}) \leftarrow \mathcal{A}_0(1^\lambda)$
$I \leftarrow \mathsf{BuildIndex}(k, D)$	$(I, st_\mathcal{S}) \leftarrow \mathcal{S}_0(\mathcal{L}_1(D))$
$(w_1, st_\mathcal{A}) \leftarrow \mathcal{A}_1(st_\mathcal{A}, I)$	$(w_1, st_\mathcal{A}) \leftarrow \mathcal{A}_1(st_\mathcal{A}, I)$
$t_1 \leftarrow \mathsf{Trpdr}(k, w_1)$	$(t_1, st_\mathcal{S}) \leftarrow \mathcal{S}_1(st_\mathcal{S}, \mathcal{L}_2(D, w_1))$
for $2 \leq i \leq q$	for $2 \leq i \leq q$
$(w_i, st_\mathcal{A}) \leftarrow \mathcal{A}_i(st_\mathcal{A}, I, t_1..t_{i-1})$	$(w_i, st_\mathcal{A}) \leftarrow \mathcal{A}_i(st_\mathcal{A}, I, t_1..t_{i-1})$
$t_i \leftarrow \mathsf{Trpdr}(k, w_i)$	$(t_i, st_\mathcal{S}) \leftarrow \mathcal{S}_i(st_\mathcal{S}, \mathcal{L}_2(D, w_1..w_i))$
let $\mathbf{t} = (t_1..t_q)$	let $\mathbf{t} = (t_1..t_q)$
output $v = (I, \mathbf{t})$ and $st_\mathcal{A}$	output $v = (I, \mathbf{t})$ and $st_\mathcal{A}$

FIGURE 2.1: SE ideal-real security game

gets access only to BuildIndex and Trpdr, since it does not know the secret key k. $st_\mathcal{A}$ is some *state* maintained by the adversary. The final *view* of $\mathcal{A}$ is the encrypted index I, and the set of generated tokens $\mathbf{t}$ and $st_\mathcal{A}$. Now observe the line correspondence between $\mathbf{Real}_{\mathsf{SE},\mathcal{A}}$ and $\mathbf{Ideal}_{\mathsf{SE},\mathcal{A},\mathcal{S}}$. In the latter, a *simulator* $\mathcal{S}$ (maintaining state $st_\mathcal{S}$) is enforced with "faking" BuildIndex and Trpdr for the same D and query keywords, *only using* leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$ (explained below). Security boils down to returning $(I, \mathbf{t}, st_\mathcal{A})$ that is distinguishable with negligible probability from the output by the real game. The challenge lies in properly using leakages $\mathcal{L}_1$ and $\mathcal{L}_2$ to create I and $\mathbf{t}$, such that (i) they "look" like those produced by real, and (ii) the Search algorithm in ideal is *consistent*, i.e., it functions similarly to that in real.

Although our schemes are independent of the underlying SE construction, as an example, we describe the leakage functions $\mathcal{L}_1, \mathcal{L}_2$ assuming the SE scheme by [6]. $\mathcal{L}_1$ is associated with what is leaked from the index alone, whereas $\mathcal{L}_2$ accounts for the leakage from the queries.

- $\mathcal{L}_1(D) = n$, where n is the size of D.
- $\mathcal{L}_2(D, W) = \langle \alpha(W), \sigma(W) \rangle$,
  where W is a set of keywords, $\alpha(W) = (id(w))_{w \in W}$ is the *access patterns*, i.e., the document ids returned by each keyword query, and $\sigma(W)$ is the *search patterns*, i.e., for every pair $w_i, w_j \in W$ such that $i \neq j$, it indicates whether $w_i = w_j$ or $w_i \neq w_j$.

2.1.2 Range Searchable Encryption (RSE) schemes

In this section we provide an overview of the prior work in privacy preserving range queries of Demertzis et al. [14] which we adequately extend so as to propose a novel technique for multidimensional range queries. At first, we present the notion of **S**ingle **R**ange **C**over (SRC) technique used in Logarithmic-SRC schemes in [14]. The idea is that the required range to

be retrieved is covered by a single range, potentially a superset of the range query. Below we present the Logarithmic-SRC schemes:

2.1.2.1 Logarithmic-SRC-i$_1$

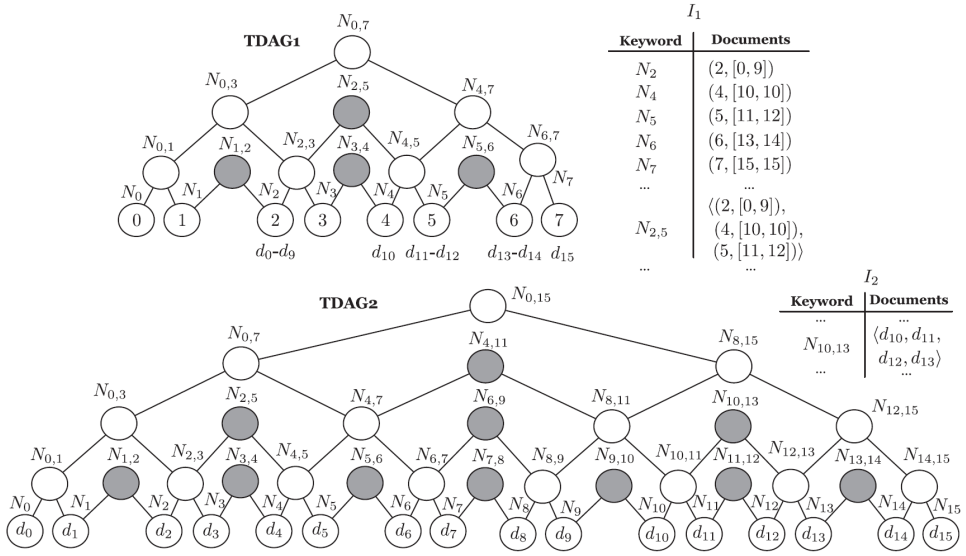


FIGURE 2.2: Double index I of SRC-i$_1$

In Logarithmic-SRC-i$_1$ there is a double index $I = (I_1, I_2)$, where I_2 indexes the tuples of the document collection D and I_1 is an auxiliary index that guides search in I_2. Index I_1 is an array based on the TDAG$_1$ structure. In order to build TDAG$_1$, we build a binary tree over domain A of the query attribute. We then inject one extra node (gray node) between every two nodes at every level of the tree, which points to the two nodes directly below it in the next level. Note that each leaf of the TDAG$_1$ holds the subset of documents that contain the value of the attribute this leaf represents. For example, in the figure above, we see that documents $\{d_0, ..., d_9\}$ have value 2 on the query attribute. Again, each father stores the info for all of their children while in the final I_1 index there is a corresponding entry for every node in TDAG$_1$. Regarding the structure of I_2 is also an array built based on TDAG$_2$ which will be described next. In I_2 for each node in TDAG$_2$ we store its document collection D', which is a subset of D. Finally, for bulding TDAG$_2$ we suppose our document collection $D = \{d_0, ..., d_{15}\}$ is sorted on the query attribute A and we construct a binary tree bottom-up. We then inject one extra node (gray node) between every two nodes at every level of the tree, which points to the two nodes directly below it in the next level.

The RSE protocol for Logarithmic-SRC-i_1 is as follows:

$k \leftarrow Setup(1^\lambda)$. Generate and output two SE keys (k_1, k_2).

$I \leftarrow BuilIndex(k, D)$. BuildIndex$(k, D)$ Build SE index I_1 on the tuple ranges using TDAG$_1$ with key k_1, and index I_2 on the sorted tuples on A using TDAG$_2$ with key k_2. Output (I_1, I_2).

$t \leftarrow Trpdr(k, w)$. This is an interactive algorithm. Parse $k = (k_1, k_2)$. Generate SE token t_1 with k_1 for the SRC node on TDAG$_1$ that covers range w, and send it to the server. Decrypt the answer to retrieve new range w . Generate SE token t_2 with k_2 for the SRC node on TDAG$_2$ that covers w , and output (t_1, t_2).

$X \leftarrow Search(t, I)$. This is an interactive algorithm. Parse $t = (t_1, t_2)$ and $I = (I_1, I_2)$. Retrieve t_1 from the owner, invoke the Search algorithm of SE on I_1 and send the result to the owner. Retrieve t_2 from the owner, invoke the Search algorithm of SE on I_2 and output the result X.

Since I_1 and I_2 are built following the construction algorithm of the underlying SE protocol and using two different keys, the leakage in each index is identical to that of the SE scheme. Therefore, having the $\mathcal{L}_1$ and $\mathcal{L}_2$ leakages of SE for both indexes and the mapping between the nodes of the first index to the nodes of the second index, we can prove the security of Logarithmic-SRC-i$_1$.

We describe the leakage of Logarithmic-SRC-i$_1$ more formally:

$-\mathcal{L}_1(D, A) = \langle m, n, n' \rangle$.

D is the dataset, A is the query attribute domain, n is the cardinality of D, n' is the cardinality of unique values of D and m is the size of A.

$-\mathcal{L}_2(D, A, W) = \langle \alpha(W), \sigma(W), (\mu(RC(w))_{TDAG_1}, \mu(RC(w'))_{TDAG_2}, uqv(RC(w)), id(RC(w'))) \rangle$.

$\alpha(W), \sigma(W)$ are the access and search patterns of the queries as defined for SE. For every query range $w \in W$, the leakage contains a tuple that consists of an alias $\mu(RC(w))_{TDAG_1}$ for the node returned by the range covering $RC(w)$ in $TDAG_1$ (similarly for $TDAG_2$, along with the unique domain values stored in $TDAG_1$ and the list of tuple IDs $id(RC(w))$ associated with the keyword $RC(w')$. It is worth mentioning that the combinations of $(\mu(RC(w))_{TDAG_1}, \mu(RC(w'))_{TDAG_2})$ with $\sigma(W)$ also leak the relation between a node in $TDAG_1$ with a node in $TDAG_2$ i.e., more than one node from $TDAG_1$ can be associated with a node in $TDAG_2$ and vice versa.

Finally, the storage overhead for the above scheme is $O(nlogm)$ while false positives and query size are $O(R + r)$.

2.1.2.2 Logarithmic-SRC-i$_2$

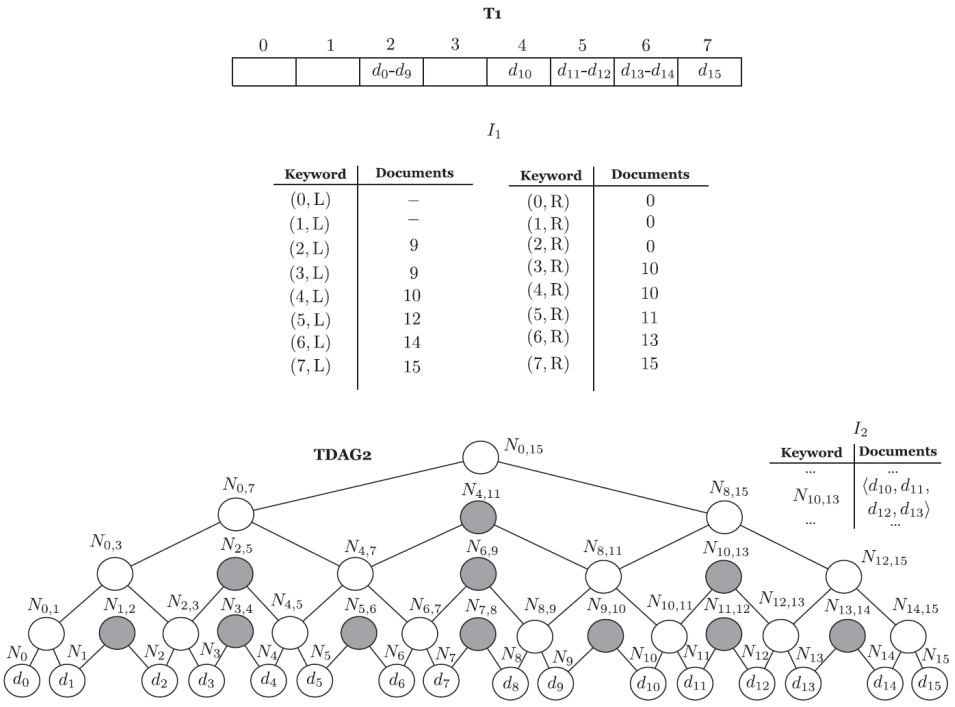


FIGURE 2.3: Double index I of SRC-i$_2$

In Logarithmic SRC-i$_2$ there is a double index $I = (I_1, I_2)$, where I_2 indexes the tuples of the document collection D and I_1 is an auxiliary index that guides the search in I_2. To begin with, I_1 index is an array where for each value in the query attribute range, we store the *first* and *last* document id that contain this value. For instance, in the example seen at the figure, in index I_1 the documents $d_0 - d_9$ have the value 2 in the query attribute.

Regarding I_2, is also an array built based on TDAG$_2$ which will be described next. In I_2 for each node in TDAG$_2$ we store its document collection D', which is a subset of D. Finally, for bulding TDAG$_2$ we suppose our document collection $D = \{d_0, ..., d_{15}\}$ is sorted on the query attribute A and we construct a binary tree bottom-up. We then inject one extra node (gray node) between every two nodes at every level of the tree, which points to the two nodes directly below it in the next level.

Let us make clear that information stored at each level of the tree is n, where n is the number of tuples. Please note, that each parent stores the document collection of its children. For example, $N_{0,1}$ is an entry in I_2 in the form of $< d_0, d_1 >$ whereas $N_{0,3}$ is $< d_0, d_1, d_2, d_3 >$.

Subsequently, we illustrate an example with one query attribute A in the where clause. Suppose, range A is 7 and the user asks for range $\alpha = \{3 - 6\}$, which states that we look for all the

documents d_i where $d_i.\alpha = 3$ || $d_i.\alpha = 4$ || $d_i.\alpha = 5$ || $d_i.\alpha = 6$. At first, by looking upon index I_1 we find that the documents to be retrieved are $D_{res} = \{d_{10} - d_{14}\}$. The single node that covers D_{res} in I_2 is $N_{8,15}$. Therefore, the result-set returned to the user is equal to $D_{final} = \{d_8 - d_{15}\}$ and the false positives are $D_{final} - D_{res} = \{d_8, d_9, d_{15}\}$.

The RSE protocol for Logarithmic-SRC-i$_2$ is as follows:

$k \leftarrow Setup(1^\lambda)$. Generate and output two SE keys (k_1, k_2).

$I \leftarrow BuilIndex(k, D)$. Build SE index I_1 using T_1 with key k_1, and index I_2 on the sorted tuples on A using TDAG$_2$ with key k_2. Output (I_1, I_2).

$t \leftarrow Trpdr(k, w)$. This is an interactive algorithm. Parse $k = (k_1, k_2)$ and range w = [i, j]. Generate SE token $t_1 = (t_{11}, t_{12})$ with k_1 for keywords (i, R) and (j, L). Decrypt the answer to retrieve new range w'. Generate SE token t_2 with k_2 for the SRC node on TDAG$_2$ that covers w', and output (t_1 ,t_2).

$X \leftarrow Search(t, I)$. This is an interactive algorithm. Parse $t = (t_1$,$t_2)$, $t_1 = (t_{11}, t_{12})$ and $I = (I_1, I_2)$. Retrieve t_1 from the owner, invoke the Search algorithm of SE for t_{11}, t_{12} on I_1 and send the results to the owner. Retrieve t_2 from the owner, invoke the Search algorithm of SE on I_2 and output the result X .

We describe the leakage of Logarithmic-SRC-i$_2$ more formally:

$-\mathcal{L}_1(D, A) = \langle m, n \rangle$.

D is the dataset, A is the query attribute domain, n is the cardinality of D and m is the size of A.

$-\mathcal{L}_2(D, A, W) = \langle \alpha(W), \sigma(W), (\mu(w_L), \mu(w_R), \mu(RC(w)), id(RC(w))) \rangle$.

$\alpha(W), \sigma(W)$ are the access and search patterns of the queries as defined for SE. For every query range $w \in W$, the leakage contains a tuple that consists of an alias $\mu(w_L)$, an alias $\mu(w_R)$ for the tokens returned by the first encrypted index, an alias $\mu(RC(w))$ for the query in $TDAG_2$, and the list of tuple ids $id(RC(w))$ associated with the keyword $(RC(w))$ in $TDAG_2$.

Finally, storage overhead for the above scheme is $O(m + nlogn)$ while false positives and query size are $O(r)$.

2.1.3 Oblivious RAM

Oblivious RAM (ORAM) introduced by Goldreich and Ostrovsky [30] is a compiler that encodes the memory such that accesses on the compiled memory do not reveal access patterns on the original memory. An ORAM scheme guarantees that there is no polynomial time adversary that can distinguish between any two sequences of accesses of the same length. The adversary can pick the initial memory and any two polynomial size sequences of accesses y_1 and y_2 of the same length ($|y_1| = |y_2|$) and by observing the oblivious accesses of $o(y_1)$ and $o(y_2)$ she will not able to distinguish them, with non-negligible probability.An ORAM scheme consists of two algorithms/protocols ORAM = (OramInitialize, OramAccess), where OramInitialize initializes the oblivious memory, and OramAccess performs the oblivious accesses.

2.1.4 Oblivious Data Structures

An oblivious data structure is a data structure that aims to hide the type and content of a sequence of operations performed on the data. Intuitively, for any two possible sequences of m operations, their resulting access patterns (i.e., the sequence of memory addresses accessed will executing the operations) must be indistinguishable. ODict offers the following protocols (see [21] for a detailed description):

- $(T, \sigma) \leftarrow$ ODictSetup$(1^\lambda, N)$: Given a security parameter λ, and an upper bound N on the number of elements, it creates an oblivious data structure T. The client sends T to the server and maintains locally the state σ.
- $(\texttt{value}, T', \sigma') \leftarrow$ ODictSearch$(\texttt{key}, T, \sigma)$: Given the search key `key` and σ, returns the corresponding value `value`, the updated T' and σ'.
- $(T', \sigma') \leftarrow$ ODictInsert$(\texttt{key}, \texttt{value}, T, \sigma)$: Given a key-value pair `key`, `value` and σ, it inserts this entry in the dictionary. It returns the updated T' and σ'.

2.1.5 Oblivious SE (OSE)

One possible way to reduce the SE query leakage would be to replace all the memory accesses performed with oblivious memory accesses using an ORAM as a black box. In that case, the only leaked information during queries is the result size, i.e., $\mathcal{L}_2(D, W) = (|\alpha(W)|)$.

2.2 Related Work

In 2000, Song et al. [4] presented the first SE scheme, secure under Chosen Plaintext Attacks (CPA). Goh [31] realized that CPA security is not adequate for the case of SE schemes. Curtmola et al. [22] introduced the state-of-the-art security definitions for SE for both, non-adaptive settings, i.e. maintaining security only if all the queries are submitted at once in one batch, as well as adaptive settings, i.e. maintaining security even if the queries are progressively submitted, and provided constructions that satisfy their definitions. The work of Curtmola et al. [22] led the way for the appearance of several new SE schemes [6, 23–26, 32–35], some of which allow updates [24–26, 34, 35], are parallelizable [25] and extend SE to support more expressive queries [5–8]. In 2014, Cash et al. [27] experimentally showed that in-memory SE cannot scale to large datasets and provided the motivation for designating locality-aware SE as a new research direction. Along this line, al. [36–39] proposed a new constructions locality-aware SE schemes.

Another important research area extends SE to support more expressive queries, such as boolean, substring, wildcard, phrase queries, range and SQL queries [5–9, 14]. The work of [9] supports a large class of SQL queries by transforming any query to selection, projection and cross product operators. However, it cannot support range and MDR queries in sub-linear time. Similarly, the works for private boolean queries [5, 6] can be used both for decomposing the range search to multi-keyword disjunctive search, and then the problem of MDR to multi-keyword conjunctive search. Unfortunately, the way that the private disjunctions and conjunctions are implemented in [5] conflicts with our performance desiderata for supporting range and MDR queries—disjunctions in [5] require significant leakages, while conjunctions require either linear search time or exponential size indexes (as we have discussed in section 1).

The works of [16, 40–43] provide privacy preserving multi-dimensional range queries for a public setting, which is different and incomparable with our approaches.

Chapter 3

Our Approach

3.1 Multi-dimensional Range Searchable Encryption

In this chapter we develop three novel encryption schemes for private multi-dimensional range queries. More specifically, we provide various extensions of Logarithmic-SRC-i$_1$ (see Section 2.1.2.1) and Logarithmic-SRC-i$_2$ schemes (see Section 2.1.2.2) that work for multiple dimensions.

3.1.1 MRSE-A

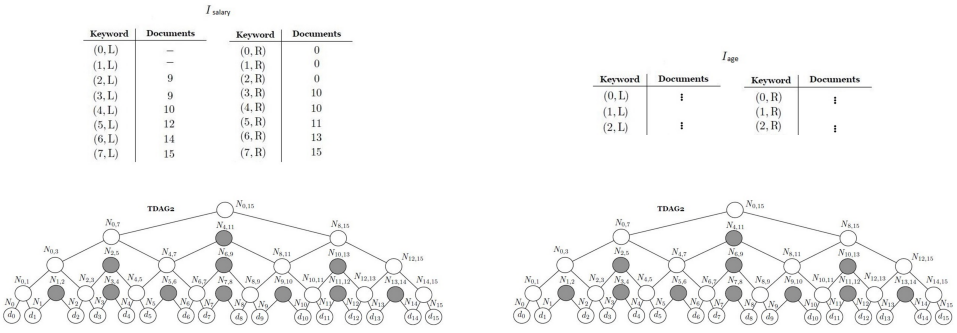


FIGURE 3.1: MRSE-A for 2 dimensions (Salary and Age)

MRSE-A is based on the Logarithmic-SRC-i$_2$ presented in Section 2.1.2.2. Recall that Logarithmic-SRC-i$_2$ has a double index $I = (I_1, I_2)$, where I_2 indexes the tuples of the document collection D and I_1 is an auxiliary index that guides the search in I_2. For d-dimensional queries, i.e., queries with d predicates in the where-clause, the proposed scheme creates d independent Logarithmic-SRC-i$_2$ instances (with d double (I_1, I_2) indexes). At first, for each dimension we use the d I_1

indexes of Logarithmic-SRC-i$_2$ in order to retrieve the number of documents satisfying the requested range for each dimension. Once all d result-sets are collected , we choose the dimension with the minimum number of tuples. We use the I_2 index of the aforementioned dimension (with the minimum number of tuples) and we retrieve the corresponding tuples. Those tuples correspond to a super set of the result, so the client has to decrypt and filter-out the false positives. Briefly, the storage for our proposed scheme is $O(dm + dn \log n)$, where m is the sum of the domain sizes of each of the d query attributes, i.e $m = \sum_{\forall i \in [1,d]} m_i$ and n is the dataset size. The false-positives are $O(min(r_i))$ for $i \in [1, d]$—namely in the worst case the number of false positives are proportional to the result size of the dimension with the smallest result set.

To further illustrate how the proposed scheme works, we present in Figure 3.1 an illustration of the encrypted indexes that MRSE-A stores for 2 dimensions (salary and age attributes), and we provide the following examples:

- $\sigma_{salary=[3K-6K]}$

  Since there is no age predicate, we use only the I_1 index for the salary attribute. Suppose that the documents to be retrieved are $D_{res} = \{d_{10} - d_{14}\}$. Using I_2 index for salary, we retrieve the node $N_{8,15}$ which contains $D_{final} = \{d_8 - d_{15}\}$.

- $\sigma_{age=[25-35]}$

  Since there is no salary predicate, we use only the I_1 index for the age attribute. Suppose that the documents to be retrieved are $D_{res} = \{d_5 - d_7\}$. Using I_2 index for age, we retrieve the node $N_{4,7}$ which contains $D_{final} = \{d_4 - d_7\}$.

- $\sigma_{salary=[3K-6K] \wedge age=[25-35]}$

  Using I_1 index for the salary attribute we learn that $|D_{salary}| = 5$ and I_1 index for the age attribute that $|D_{age}| = 3$. Since, D_{age} has the minimum number of tuples we use I_2 index for age attribute and we retrieve the node $N_{4,7}$.

For the MRSE-A given the number of distinct query attributes d, i.e., $l \in \{1, 2, \cdots, d\}$, we assume that the client stores the $\{min, max\}$ values for each attribute along with n, which represents the total number of tuples. We refer to the min value of a query attribute l as $min.l$, and to the max value as $max.l$. Similarly, we denote with $I_1.l$ the I_1 index of attribute l. Finally, we denote with $t2.min$ the trapdoor for $I_{2,min}$ index of attribute min, i.e., the attribute whose result size retrieved from I_1 is the minimum. In particular:

$k \leftarrow Setup(1^\lambda)$. Generate and output d pairs of SE keys $(k_1.l,\ k_2.l)$, one pair for each attribute.

$I \leftarrow BuilIndex(k, D)$. For each $l \in [1, d]$, build the SE indexes $I_1.l$ using $T_1.l$ (as in SRC-i_2) with key $k_1.l$ and index $I_2.l$ on the sorted tuples of $A.l$ using TDAG_2 with key $k_2.l$, and output $(I_1.l, I_2.l)$.

$t \leftarrow Trpdr(k, w)$. This is an interactive algorithm which runs in 2 rounds. Round 1: For $l \in [1, d]$ parse $k.j$ as $(k_1.l,\ k_2.l)$ and range $w.l$ as $[i.l, j.l]$. Generate SE tokens $t_1.l = (t_{11}.l,\ t_{12}.l)$ with $k_1.l$ for keywords $(i.l, R.l)$ and $(j.l, L.l)$. These tokens will be used as inputs to multiple calls of the $Search$ algorithm below. Round 2: After retrieving/decrypting the answers of all the above executions of the $Search$ algorithms the client can identify the dimension min with the minimum result size (as described above) and its corresponding range w'. Finally, the client generates a SE token $t_2.min$ with $k_2.min$ for the SRC node of $\text{TDAG}_2.min$ that covers the range w', and output $(\boldsymbol{t_1}, t_2)$.

$X \leftarrow Search(t, I)$. This is an interactive algorithm which runs in 2 rounds. Round 1: Parse $t.l = (t_1.l\ , t_2.l)$, $t_1.l = (t_{11}.l,\ t_{12}.l)$ and $I = (I_1.l,\ I_{2,min})$. Retrieve $t_1.l$ from the owner, invoke the Search algorithm of SE for $t_{11}.l, t_{12}.l$ on $I_1.l$ and send the results to the owner. Round 2: Retrieve $t_2.min$ from the owner (second round of the $Token$ algorithm), invoke the Search algorithm of SE on $I_{2,min}$ and output the result X.

We describe the leakages of MRSE-A more formally:

$-\mathcal{L}_1(D, A) = \langle m, n, d \rangle$.

D is the dataset, A is a vector of the domain sizes of the d query attributes, n is the cardinality of D, m is the sum of the domain sizes of the query attributes and d is the number of the distinct attributes in the where-clause of the query.

$-\mathcal{L}_2(D, A, W) = \langle \sigma(W), \mu(w_{i,L}), \mu(w_{i,R}))_{\forall i \in [1,d]}, \mu(RC(w_{min})), id(RC(w_{min})) \rangle$.

W corresponds to the multi-dimensional range query and is in the form of $w = (w_1, w_2, ..., w_d)$, where w_i is the range for the i-th attribute ($i \in [1, d]$) in the where-clause of the query. The leakage contains $\sigma(W)$, which is the search patterns of $w_1, w_2, ..., w_d$; for each w_i a tuple that consists of an alias $\mu(w_{i,L})$ and an alias $\mu(w_{i,R})$ for the tokens returned by the first encrypted indexes; it contains an alias $\mu(RC(w_{min}))$ in $I_{2,min}$ of the attribute whose result set is the minimum, and the list of tuple ids $id(RC(w_{min}))$ associated with the keyword $RC(w_{min})$ in $I_{2,min}$.

Qualitative Comparison. MRSE-A is the simplest and most intuitive way to extend Logarithmic-SRC-i_2 to work for multiple dimensions. The main drawbacks of this approach are (i) the increased storage demands, (ii) the search time/false positives which are linear to the number of returned tuples for the dimension with the minimum result size, i.e., there is case that the result size is empty and $O(min(r_i))$ is proportional to $O(n)$.

3.1.2 MRSE-B

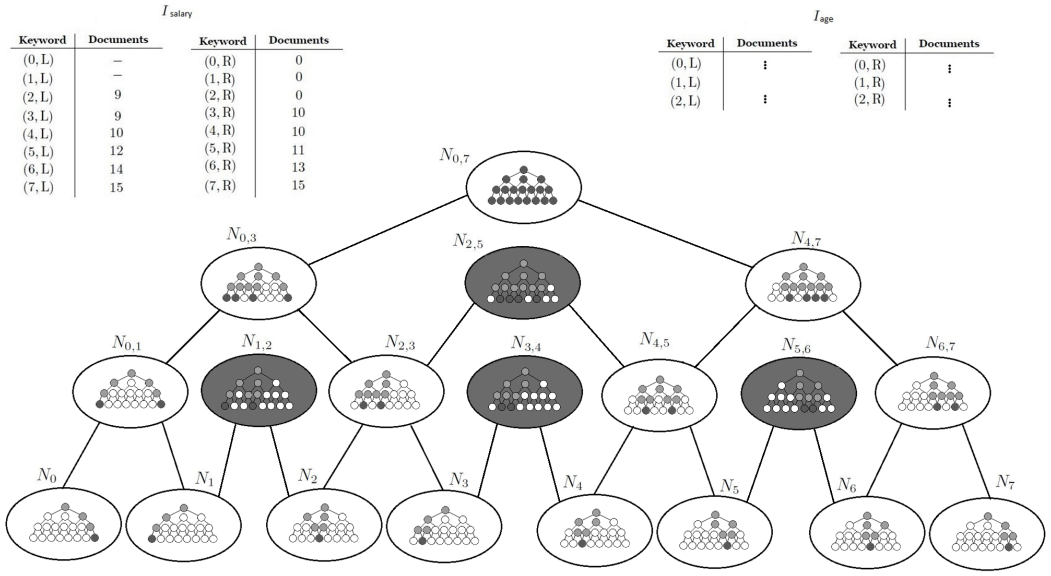


FIGURE 3.2: MRSE-B for 2 dimensions (Salary and Age)

MRSE-B is based on Logarithmic-SRC-i$_2$ presented in Section 2.1.2.2. In particular, given a document collection D and d query attributes, we create d indexes of type I_1 (see Logarithmic-SRC-i$_2$) which are auxiliary indexes that guide the search in the index I_2. The difference now lies on the I_2 index in which each node contains a nested I_2 index for the next dimension. The composite I_2 is stored as an array, where entries are the concatenation of the outer node with the corresponding inner node. The I_2 index of MRSE-B is based on a d-dimensional TDAG$_2$.

For example, we provide in Figure 3.2 the indexes of MRSE-B for 2 attributes—salary and age attribute, assuming that the outer nodes correspond to the sorted salary attribute and the inner nodes to the sorted age attribute.

As an example, consider the node $N_{0,3}$ of I_2 index for salary which points to the subset $D' = \{d_0, ..., d_3\}$. Now, D is sorted by age attribute and any document $d_i \notin D'$ becomes a *null* info

leaf in the inner TDAG$_2$. Note that the order of documents change, because of the different sorting, which means the outer d_0 might become the inner d_7.

To further illustrate how MRSE-B works, we consider the following examples:

- $\sigma_{salary=[3K-6K]}$

  Using I_1 index for the salary attribute, suppose that the documents to be retrieved are $D_{salary} = \{d_3, d_4, d_5\}$. Since there is no age predicate, we retrieve the node $N_{2,5}N_{0,7}$, i.e., the $N_{0,7}$ node of the inner TDAG$_2$ from $N_{2,5}$ node of the outer TDAG$_2$.

- $\sigma_{age=[25-35]}$

  Using I_1 index for the age attribute, suppose that the documents to be retrieved are $D_{age} = \{d_1 - d_2\}$. Since there is no salary predicate, we retrieve the node $N_{0,7}N_{1,2}$ of the I_2 index, i.e., the inner $N_{1,2}$ node of the outer $N_{0,7}$ node.

- $\sigma_{salary=[3K-6K] \wedge age=[25-35]}$

  Using I_1 index for salary attribute, and I_1 index for age attribute we learn the boundaries of the second query in I_2 index—assuming that those boundaries are $[2, 5]$ for salary attribute and $[1, 2]$ for the age attribute we retrieve the node $N_{2,5}N_{1,2}$ of the I_2 index, i.e., the $N_{1,2}$ inner node from the $N_{2,5}$ outer node.

We provide the MRSE-B scheme below:

$k \leftarrow Setup(1^\lambda)$. Generate and output SE keys $k_1.l$ one for each attribute and k_2 for I_2, which now is the same for all dimensions.

$I \leftarrow BuilIndex(k, D)$. For each $l \in [1, d]$, build SE index $I_1.l$ using $T_1.l$ (as in SRC-i$_2$) with key $k_1.l$ and index I_2 using the d-dimensional TDAG$_2$ with key k_2, which is properly sorted as we describe above. Output $(I_1.l, I_2)$.

$t \leftarrow Trpdr(k, w)$. This is an interactive algorithm which runs in 2 rounds. Round 1: For each $l \in [1, d]$, parse $k = (k_1.l, k_2)$ and range $w.l = [i.l, j.l]$ and generate SE tokens $t_1.l = (t_{11}.l, t_{12}.l)$ with $k_1.l$ for keywords $(i.l, R.l)$ and $(j.l, L.l)$. Round 2: Decrypt the answers to retrieve the new range w'. Generate SE token t_2 with k_2 for the SRC node on the d-dimensional TDAG$_2$ that covers w', and output $(\boldsymbol{t_1}, t_2)$.

$X \leftarrow Search(t, I)$. This is an interactive algorithm which runs in 2 rounds. Parse $t = (t_1.l, t_2)$, $t_1.l = (t_{11}.l, t_{12}.l)$ and $I = (I_1.l, I_2)$. Round 1: Retrieve $t_1.l$ from the owner,

invoke the Search algorithm of SE for $t_{11}.l, t_{12}.l$ on $I_1.l$ and send the results to the owner. Round 2: Retrieve $t_2 = t_1.1||t_1.2||...||t_1.d$ from the owner, invoke the Search algorithm of SE on I_2 and output the result X.

We describe the leakages of MRSE-B more formally:

$-\mathcal{L}_1(D, A) = \langle m, n, d \rangle$.

D is the dataset, A is a vector of the domain sizes of the d query attributes, n is the cardinality of D, m is the sum of the domain sizes of the query attributes and d is the number of the distinct attributes in the where-clause of the query.

$-\mathcal{L}_2(D, A, W) = \langle \sigma(W), \mu(w_{i,L}), \mu(w_{i,R}))_{\forall i \in [1,d]}, \mu(RC(w')), id(RC(w')) \rangle$.

W corresponds to the multi-dimensional range query and is in the form of $w = (w_1, w_2, ..., w_d)$, where w_i is the range for the i-th attribute ($i \in [1, d]$) in the where-clause of the query. The leakage contains: $\sigma(W)$ which is the search patterns of $w_1, w_2, ..., w_d$; for each w_i a tuple that consists of an alias $\mu(w_{i,L})$ and an alias $\mu(w_{i,R})$ for the tokens returned by the first encrypted indexes; for the new request range w' it contains an alias $\mu(RC(w'))$ in I_2 and the list of tuple ids $id(RC(w'))$ associated with the keyword $RC(w')$ in I_2.

Finally, the storage for the MRSE-B is $O(dm + dlog^d n)$ and the false positives are $O(min(r_i))$.

Qualitative Comparison. We observe that MRSE-B has asymptotically the same false positive bound with MRSE-A; however we observe in Section 4 that MRSE-B significantly reduces the false positives compared with MRSE-A scheme. In addition, MRSE-B has better leakage profile compared with MRSE-A. The main drawbacks of MRSE-B are (i) that MRSE-B asymptotically increases the index size, and (ii) still the search time/false postives are not bounded by the actual result size.

3.1.3 MRSE-C

MRSE-C is based on the Logarithmic-SRC-i$_1$ presented in Section 2.1.2.1. For the d-dimensional query setting, define as $A = \{\alpha_1, \alpha_2, \cdots, \alpha_d\}$ the sequence of attributes in the order that appear in a query. For the MRSE-C scheme we have a double index $I = (I_1, I_2)$ where I_1 is a TDAG$_1$ based index built over the domain of the α_1 attribute. I_2 index follows the structure of TDAG$_2$ and is built based on the sorted document collection D on α_1 attribute while each of its nodes is a chain of d nested (I_1, I_2) pairs. Note that each node in the initial I_2 index *spans*/contains a subset of the original document collection. So at each node using only this subset of documents,

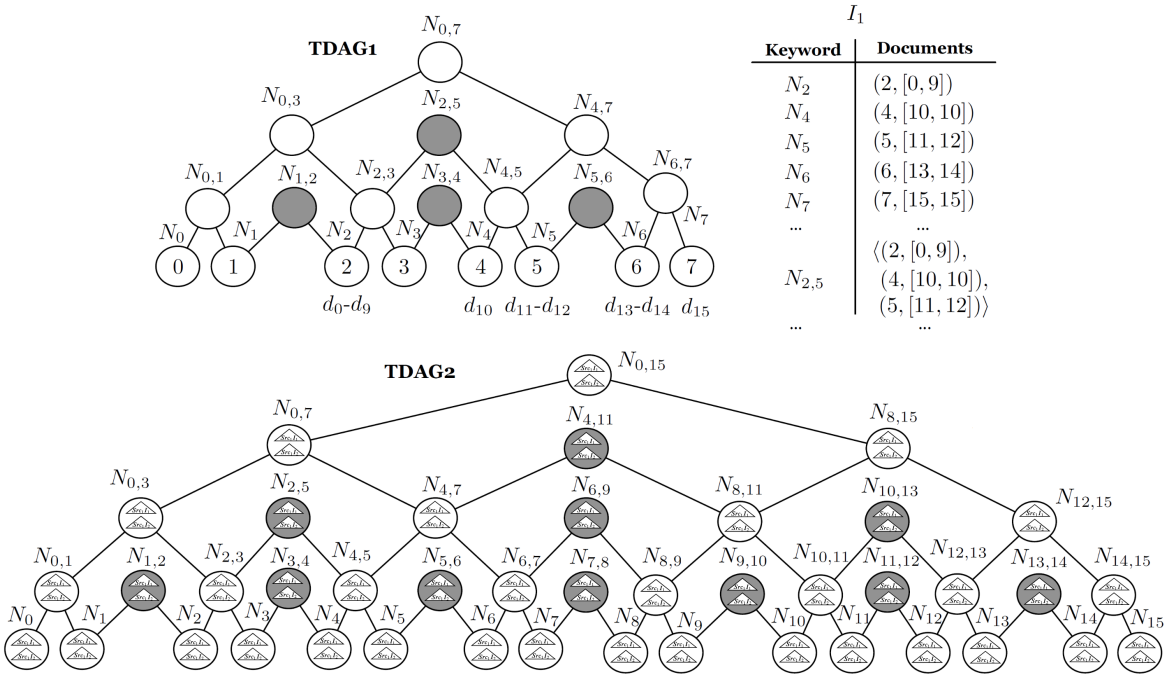


FIGURE 3.3: MRSE-C for 2 dimensions (Salary and Age)

we store a new pair of I_1 and I_2 indexes that are built upon α_2 attribute. Then we apply the same procedure recursively; for each node of this (I_1, I_2) pair we build a new set of indexes based on the corresponding subset of the document collection D and α_3 attribute. The same follows until we reach α_d attribute. The procedure explained above in a node-level is applied for every node of the initial I_2 index.

For example, we provide in Figure 3.3 the new index of MRSE-C for 2 attributes—salary and age attribute. We consider the following order for those 2 attributes: $A = \{salary, age\}$. Thus, we build a I_1 index over the salary domain and a I_2 index over the sorted document collection on salary. Then we build new pairs of $I_{age} = \{I_1, I_2\}$ double indexes; one for each node of salary's I_2 index. We omit in Figure 3.3 the index I_1 of the salary attribute.

To give a more concrete description, observe that the root of I_2 index for the salary attribute points to the document collection D. Therefore, the new index $I_{age} = \{I_1, I_2\}$ is built over D accordingly. On the other hand, in any other node we process only the subset of documents that corresponds to it. Let us use node $N_{0,3}$ of I_2 for salary as an example. Observe that the node contains $D' = \{d_0, ..., d_3\} \subseteq D$ which means that I_{age} is built only using D'.

To further illustrate how MRSE-C works, we consider the following examples:

- $\sigma_{salary=[3K-6K]}$

Using I_1 index for the salary attribute, we get that $D_{res_1} = \{d_3, d_4, d_5\}$, which is best covered by $N_{2,5}$ node. Since age is set to ALL, using I_1 index for the age attribute of $N_{2,5}$ node, we get $N_{0,15}$ node. Using I_2, we retrieve the node $N_{2,5}N_{0,15}$.

- $\sigma_{age=[25-35]}$

  Using I_1 index for the salary attribute witch is set to ALL, we get the $N_{0,16}$ node. Then, using I_1 index for the age attribute of $N_{0,15}$ node, we get that $D_{res_2} = \{d_{10} - d_{14}\}$ which is best covered by $N_{8,15}$ node. Using I_2, we retrieve the node $N_{0,15}N_{8,15}$.

- $\sigma_{salary=[3K-6K] \wedge age=[25-35]}$

  Using I_1 index for the salary attribute, we get that $D_{res_1} = \{d_3, d_4, d_5\}$, which is best covered by $N_{2,5}$ node. Then, using I_1 index for the age attribute of $N_{2,5}$ node, we get that $D_{res_2} = \{d_{10} - d_{14}\}$ which is best covered by $N_{8,15}$ node. Using I_2, we retrieve the node $N_{2,5}N_{8,15}$.

We provide the MRSE-C scheme below:

$k \leftarrow Setup(1^\lambda)$. Generate and output SE keys $k_1.l$ one for each attribute and k_2 for I_2, which now is unique for all dimensions.

$I \leftarrow BuilIndex(k, D)$. For each $l \in [1, d]$, build SE index $I_1.l$ using $T_1.l$ (as in SRC-i$_2$) with key $k_1.l$ and index I_2 using a d-dimensional TDAG$_2$ as described above with key k_2. Output $(I_1.l, I_2)$.

$t \leftarrow Trpdr(k, w, i)$. This is an $d + 1$-interactive algorithm where i denotes the current round and takes values in the range $[1, \ldots, d + 1]$. Round $i \leq d$: Parse $k.i$ and range $w.i$. Generate SE token $t_1.i$ with $k_1.i$ for the SRC node in current TDAG$_1$ that covers $w.i$ and send the answer to the server. Round $i = d + 1$: Decrypt the previous answer (for $i = d$) to retrieve the new range $w'.d$. Generate SE token t_2 with k_2 for the SRC node on TDAG$_2$ that covers $w'.d$, and output $(\boldsymbol{t_1}, t_2)$.

$X \leftarrow Search(t, I)$. This is an $d + 1$-interactive algorithm. Parse $t = (t_1.l, t_2)$ and $I = (I_1.l, I_2)$. Round 1: Retrieve $t_1.1$ from the owner, invoke the Search algorithm of SE on $I_1.1$ and send the results to the owner. Round $i \leq d$: Retrieve $t_1.i$ which depends on the previous search (for $i - 1$), invoke the Search algorithm of SE on $I_1.i$ and send the results to the owner. Round $d + 1$: Retrieve t_2 from the owner, invoke the Search algorithm of SE on I_2 and output the result X.

We describe the leakages of MRSE-C more formally:

$-\mathcal{L}_1(D, A) = \langle m, n, n', d \rangle$.

D is the dataset, A is a vector of the domain sizes of the d query attributes, n is the cardinality of D and m is the sum of the domain sizes of the query attributes. In addition, n' is a vector with the unique values for the d attributes ($n' = [n'_1, n'_2, ..., n'_d]$) and d is the number of the distinct attributes in the where-clause of the query.

$-\mathcal{L}_2(D, A, W) = \langle \sigma(W), (\mu_{TDAG_1}(RC(w_i)))_{\forall i \in [1,d]}, \mu_{TDAG_2}(RC(w')), (uqv(RC(w_i)))_{\forall i \in [1,d]}, id(RC(w')) \rangle$.

W corresponds to the multi-dimensional range query and is in the form of $w = (w_1, w_2, ..., w_d)$, where w_i is the range for the i-th attribute ($i \in [1, d]$) in the where-clause of the query. The leakage contains $\sigma(W)$, which is the search patterns of $w_1, w_2, ..., w_d$. For each w_i the leakage contains a tuple that consists of an alias $\mu_{TDAG_1}(RC(w_i))$ for the node returned by the range covering $RC(w_i)$ in $I_1.i$ along with the distinct values of the range w_i. It also contains an alias $\mu_{TDAG_2}(RC(w'))$ for the final query in the index I_2 and the list of tuple IDs $id(RC(w_i))$ associated with the keyword $RC(w')$

Qualitative Comparison. We observe that MRSE-C can asymptotically bound the number of false positives to be proportional to actual result and range size at the expense of more leakage and increased number of interactions. The main drawbacks of MRSE-C are (i) that MRSE-C asymptotically increases the index size, (ii) d interactions are required, and (iii) the d interactions introduce more leakage than MRSE-A and MRSE-B schemes.

3.1.4 MRSE Schemes With New Trade-offs Between Index Size and False Positives

The main drawback of the MRSE schemes described in Sections 3.1.1-3.1.3 is the increased size of the encrypted indexes. We describe an approach to address the aforementioned limitation by introducing a new trade-off between space and false-positives. The idea is that instead of storing all the levels for the indexes in MRSE-A,MRSE-B and MRSE-C, we only we store s evenly distributed levels for each dimension; the missing levels increase the number of false positives. In particular, for the MRSE-A and MRSE-B schemes we only apply this technique for the I_2 indexes for which we store only s levels since the space for I_1 is linear in m. Finally for MRSE-C, we keep only s levels both for I_1 and I_2 indexes. As we mentioned above, this idea introduces more false positives; we observe that the number of false positives will be increased by a factor of $2^{\frac{logn}{s}} = n^{1/s}$.

3.2 Oblivious MRSE (OMRSE)

OMRSE scheme creates an oblivious searchable R-tree. In particular, we create a plaintext R-tree, and we store each node in a non-recursive Path-ORAM [11] (P1). For each node we choose at random a leaf i (in P1) and we insert the node in the chosen leaf. In order to be able to traverse the oblivious R-tree, we organize each R-tree node in the following way:

$$((id, data), childrenID, childrenPos)$$

id is the key for the searches in the R-tree, $data$ corresponds to the actual data of the corresponding plaintext node of the R-tree, $childrenID$ is an array which stores the ids for the node's children and $childrenPos$ is an array which stores the P1 leaf numbers for the node's children.

The client only stores the position of the R-tree's root in order to be able to obliviously search the R-tree. For a given query, we first retrieve and decrypt the root, based on the root's $data$ we know in which of root's children nodes we have to continue the search; having their ids and their positions in P1 we are able to continue the search until we retrieve the result of the requested query.

After a search query, we need to store all the "touched" nodes to new positions in P1 (choosing at random for each touched node a new leaf). We perform the re-mapping in a bottom-up manner—starting from the leaves of the R-tree and continuing to their parents until we reach the root node. The above traversal guarantees that when a parent node have to be placed in P1 the new locations of its children have been determined.

Finally, the storage for the OMRSE is $O(n)$ and search efficiency $O(t(n) \cdot \log^2 n)$ assuming that the search complexity for the plaintext R-tree is $t(n)$.

We describe the leakages of OMRSE more formally:

- $\mathcal{L}_1(D, A) = \langle n \rangle$.

  n is the cardinality of D.

- $\mathcal{L}_2(D, A, W) = \langle \phi(W) \rangle$.

  W corresponds to the multi-dimensional range query and is in the form of $w = (w_1, w_2, ..., w_d)$, where w_i is the range for the i-th attribute ($i \in [1, d]$) in the where-clause of the query. $\phi(W)$ denotes the number of "touched" nodes in the plaintext R-tree for answering the query W.

Qualitative Comparison. We observe that OMRSE significantly reduces both $\mathcal{L}_1$ and $\mathcal{L}_2$ leakages. OMRSE is the first scheme that requires linear index size (MRSE-A, MRSE-B, MRSE-C schemes require super-linear index size). The search complexity of OMRSE depends on the search complexity of the plaintext R-tree. The worst case search efficiency of OMRSE is the same with MRSE-A, MRSE-B, MRSE-C when s is tuned to be 1 (choosing the index size in all of these schemes to be similar).

Chapter 4

Experimental Evaluation

4.1 Setup

We implemented our solutions in C++ and conducted our experiments on a 64-bit machine with an Intel® CoreTM i7-3540M CPU at 3.0GHz and 16GB RAM, running Linux Ubuntu 16.08. We utilized the OpenSSL library for the entailed cryptographic operations and we used AES128-CBC for encryption. In addition, we experimented both with a real-world dataset and a synthetic one. The real- world dataset is from the U.S. Postal Service (www.app.com), called USPS, and contains almost $400K$ employee records. The query attributes of the USPS dataset we used in our experiments are the annual salary field with domain $A_1 = \{0, ..., 276840\}$, fund name with domain $A_2 = \{0, ..., 16\}$, country name with domain $A_3 = \{0, ..., 60\}$ and total salary with domain $A_4 = \{0, ..., 392877\}$.

We also generated a synthetic dataset with 4 query attributes; salary, age, dept_code and monthly hours. The domain for the salary attribute, which follows a skewed distribution with mode equal to $15,000$, is $A_1 = \{0, 1, ..., 300,000\}$. Next, we use a skewed distribution with mean equal to mean 40, standard deviation 36, skew equal to 0.3 and kurtosis kurt equal to 3 for the age attribute with domain $A_2 = \{18, 19, ..., 75\}$. For the dept_code attribute with domain $A_3 = \{100, ..., 600\}$ we use a uniform distribution while for the monthly hours attribute with domain $A_4 = \{0, ..., 300\}$ we use a gaussian distribution with mean equal to 150 and standard deviation equal to 22.

4.2 False Positives/Search Time

For the evaluation of the proposed schemes, we first conduct a series of experiments so as to observe their behavior with regards to the result sizes retrieved for a set of random queries. In particular, we are interested in observing how many more results each of the schemes returns in comparison to the true number of tuples that answer a given query.

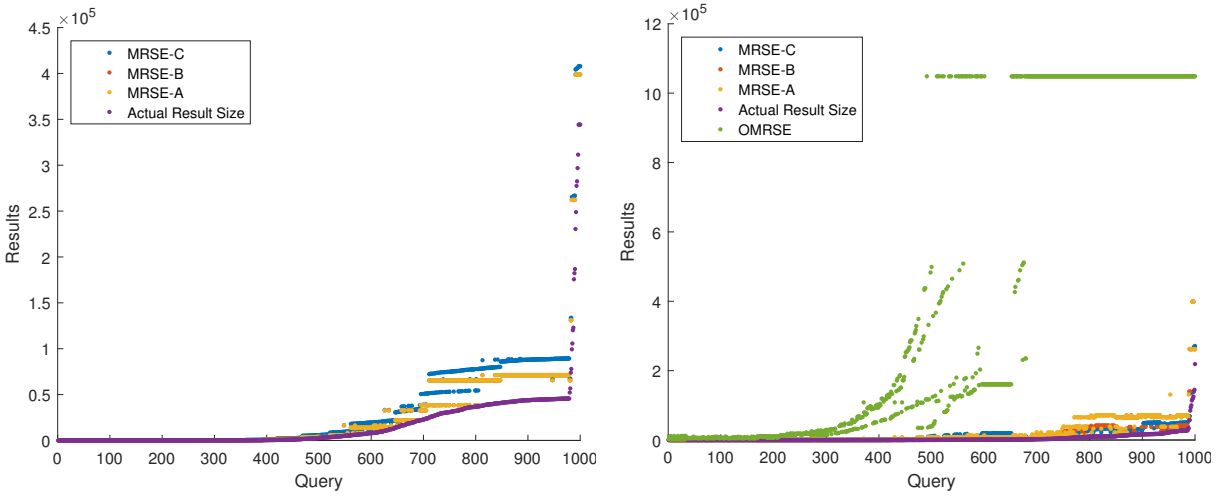
To begin with, in figures 4.1, 4.2 and 4.3 we provide the experimental results using a random set of queries answered by 20%, 40% or 60% respectively of the domain of each attribute in the query while the dimension of the queries varies from 1 up to 4. From the theoretical analysis of our schemes, we expect that MRSE-A will be the one with the worst false positive behavior since the result size always matches the size of the smallest result set among the d results sets. However, for MSRE-B although in the worst case the false positives could match those of MRSE-A, in the average case it improves the false positives. The upper bound for MRSE-B can be reached in the case when a query is answered from nodes that are closer to the root; the closer to the root a node is, the more tuples it *spans*. From the experimental evaluation, we observe that indeed MRSE-B reduces the false positives of MRSE-A while in the special case of 1-dimensional queries the two schemes are identical(since the I_1, I_2 indexes have the same structure). Subsequently, for MRSE-C we observe that its false positives outperform those of MRSE-A and MRSE-B while we increase the number of dimensions and the percentage of the tuples that answer a query per dimension. When the actual result size increases both MRSE-A and MRSE-B will pick a node closer to the root which *spans* in the worst case the whole database. However, by construction of the I_2 index the corresponding node in MRSE-C contains only the proper subset of documents that answer the query. Finally, the perfomance of OMRSE when the indexes of MRSE schemes are fully stored cannot compete any of the MRSE schemes. Nonetheless, as we will discuss in section 4.4 when we reduce the size of the indexes of MRSE schemes so that it is comparable to the size of the index in OMRSE, the latter starts to competitively compare with the former schemes for small range sizes and dimensions. In other words, we observe a trade-off between storage and false positive between OMRSE and MRSE protocols. To sum up, MRSE-B experimentally reduces the false positives of MRSE-A while MRSE-C, which is the only scheme that bounds the number of false positives to be proportional to the actual result and range size, performs best when d and range size get larger.

4.3 Index Size

In Figure 4.4, we observe how the memory space required to store the indexes of each of the MRSE increases with the respect to the size of the original database for different numbers of dimensions d that vary from 1 up to 4. In the special case of 1-dimensional queries, MRSE-A and MRSE-B are identical since their double index $I = (I_1, I_2)$ is the same but as the number of dimensions increases the storage required for MRSE-A is less as expected from the theoretical analysis. MRSE-C is the scheme that requires the more space, although it may not seem obvious from the analysis where constants do not appear. Finally, OMRSE as explained before is the one that requires the least space of all.

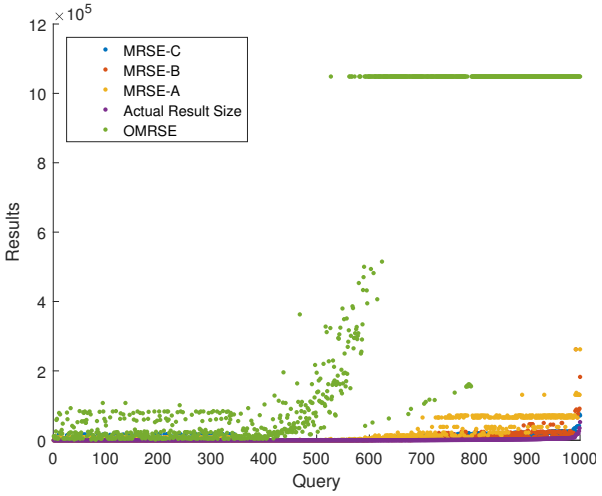
4.4 Index Size and False Positives/Search Time for s=2 Levels

In this section, we compare OMRSE and MRSE schemes under the same memory configuration. In order to reduce the space for the MRSE schemes, we store only $s = 2$ levels for each dimension of the original structure. Under this setting, as presented in figures 4.5, 4.6, 4.7, OMRSE is competitively better(up to 100× speed-up) than MRSE schemes for small range sizes and small number of dimensions while it can be up to 2× worse when the range sizes and dimensions increase. However, the trade-off between storage and false positive rate between the OMRSE and the MRSE schemes is significant since OMRSE generally requires less space than the other schemes, even when we store $s = 2$ levels per dimension. All in all, OMRSE does not always outperform the MRSE schemes even comparable memory configurations but the sacrifice in false positives performance is significantly comparable with the gain in storage.

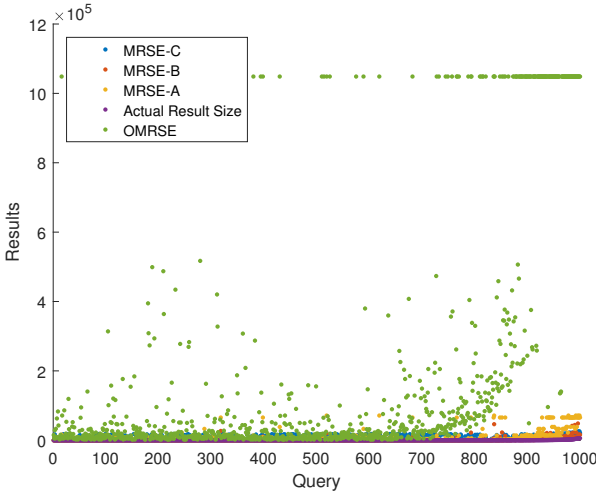


(A) 1-Dimension

(B) 2-Dimensions

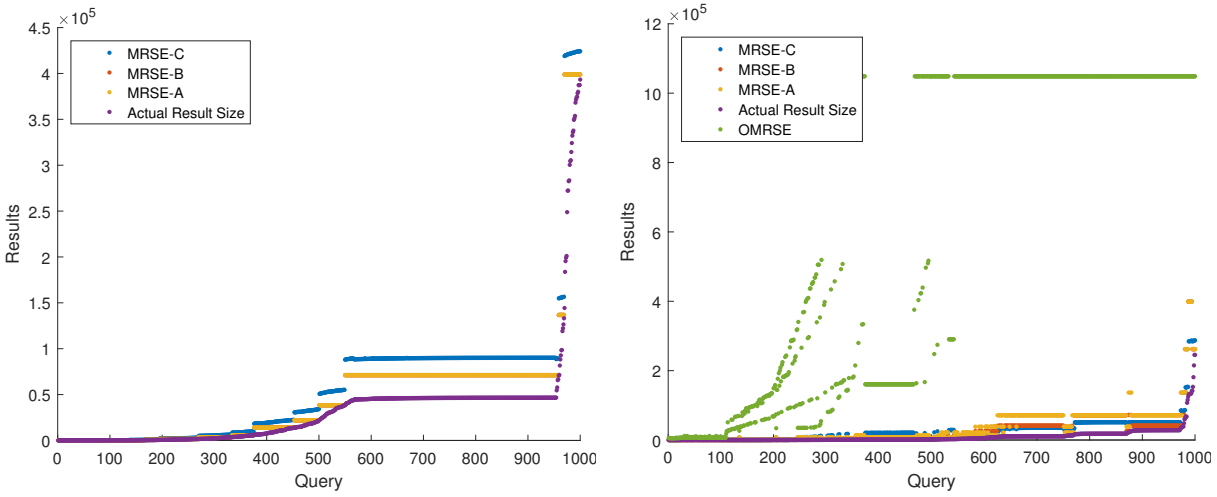


(C) 3-Dimensions



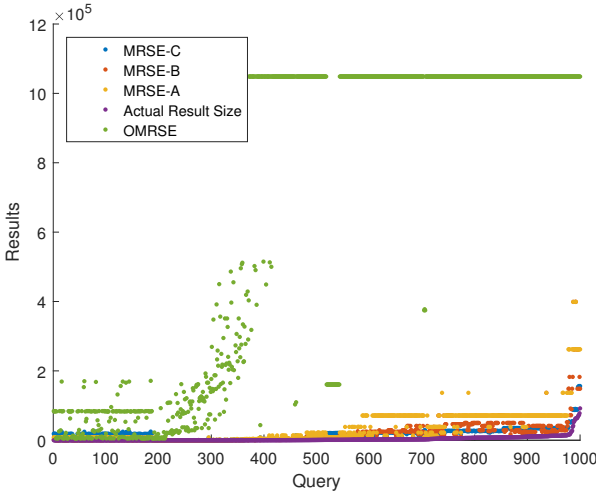
(D) 4-Dimensions

FIGURE 4.1: 20% Range Query Size for each dimension (s=ALL)

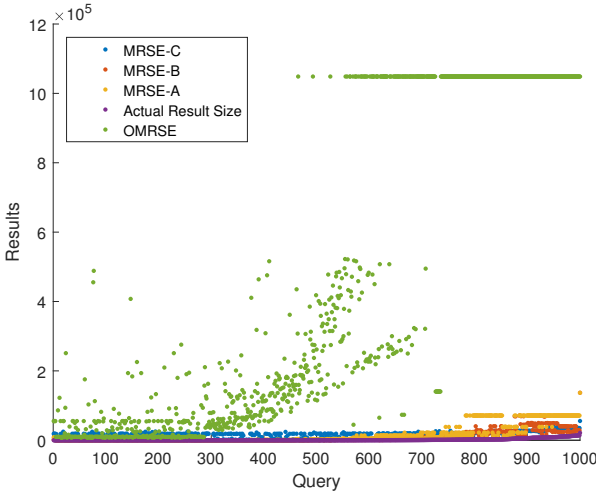


(A) 1-Dimension

(B) 2-Dimensions

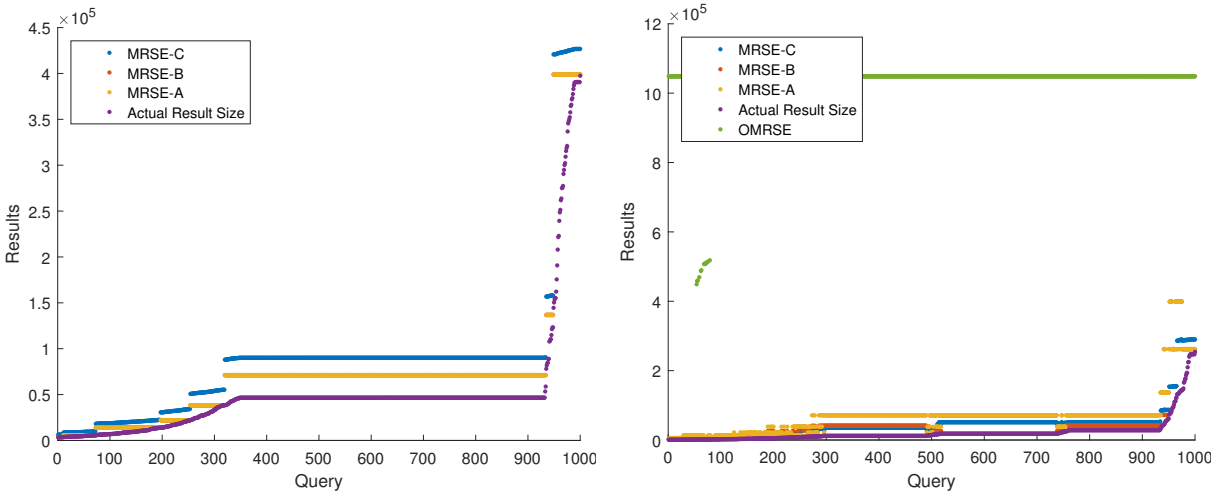


(C) 3-Dimensions



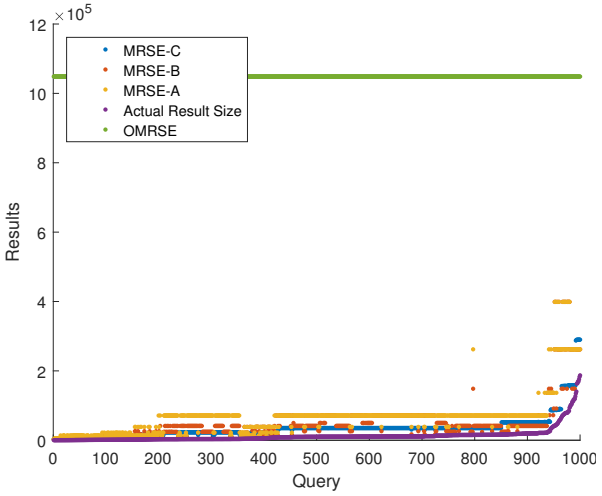
(D) 4-Dimensions

FIGURE 4.2: 40% Range Query Size for each dimension (s=ALL)

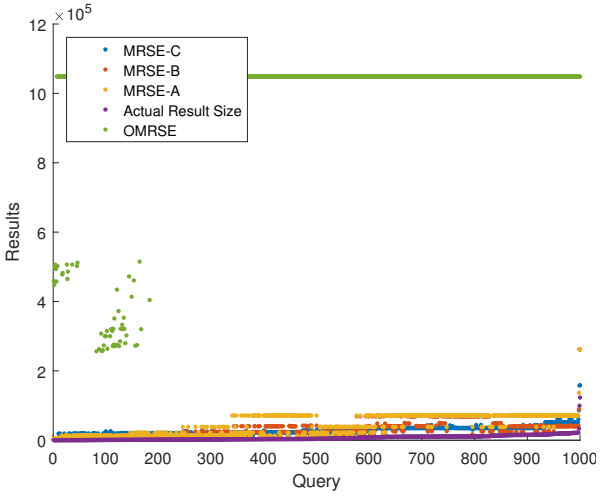


(A) 1-Dimension

(B) 2-Dimensions

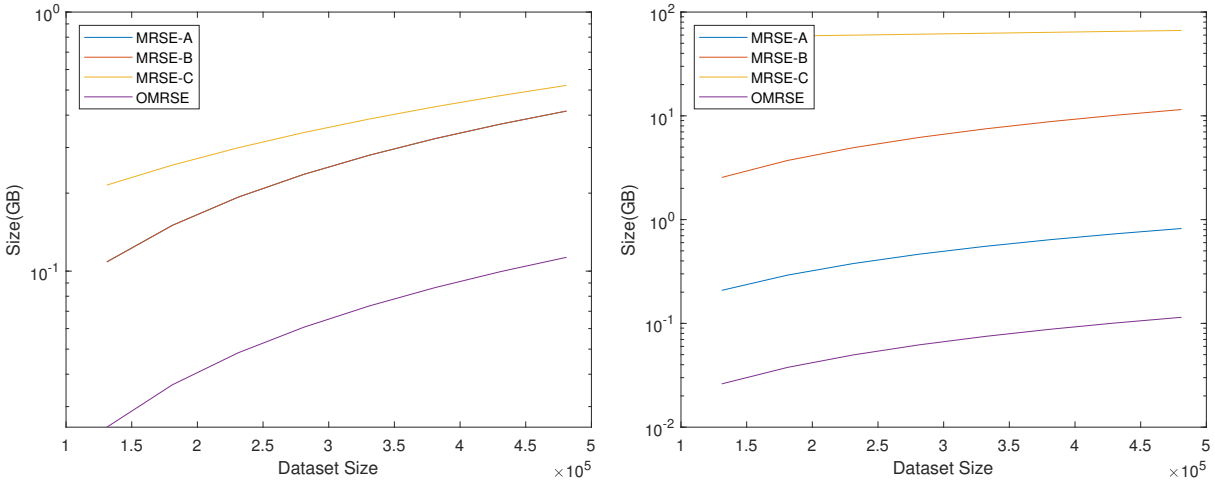


(C) 3-Dimensions



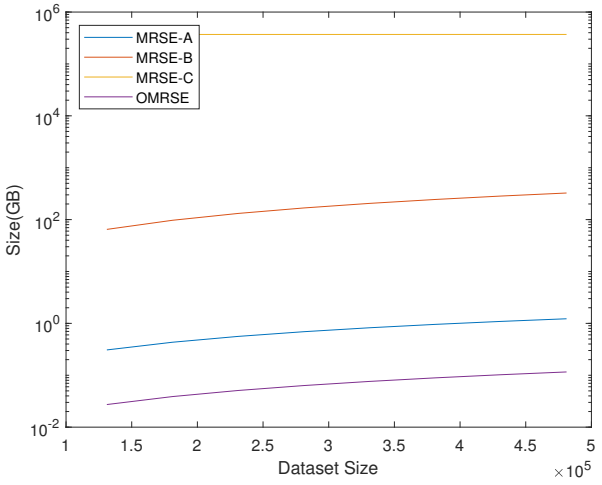
(D) 4-Dimensions

FIGURE 4.3: 60% Range Query Size for each dimension (s=ALL)

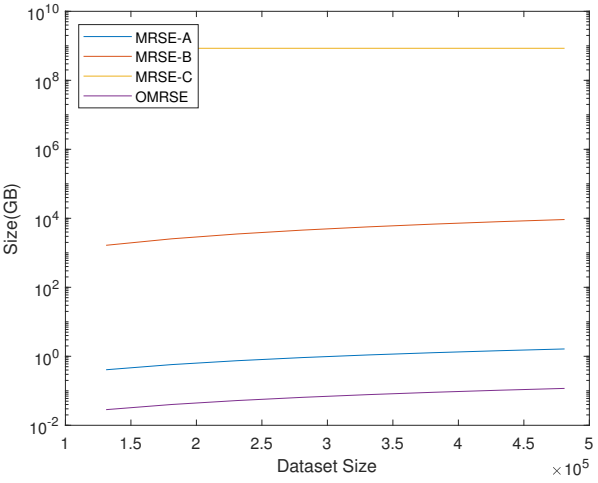


(A) 1-Dimension

(B) 2-Dimensions

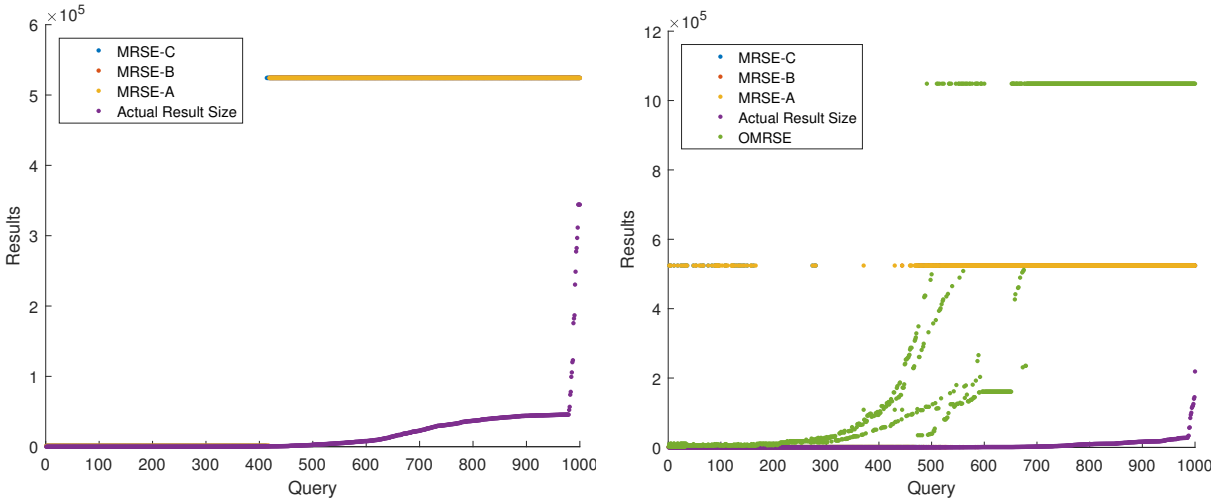


(C) 3-Dimensions



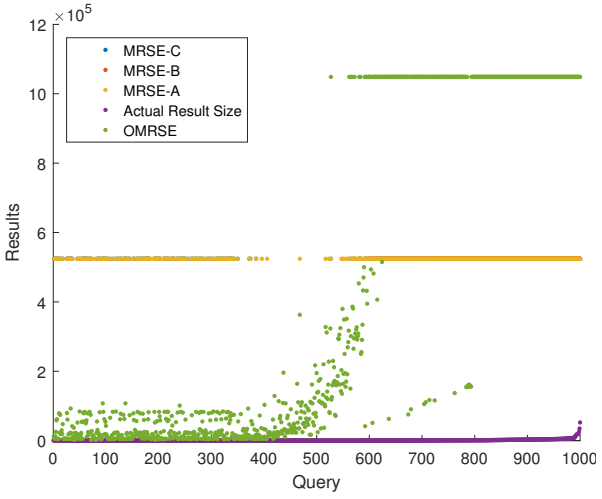
(D) 4-Dimensions

FIGURE 4.4: Index Size (s=ALL)

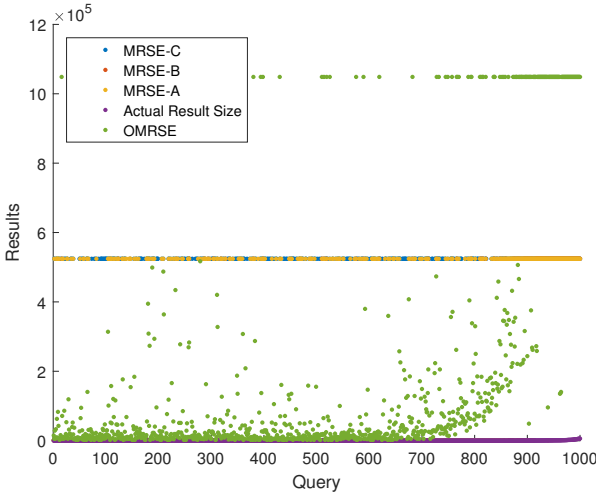


(A) 1-Dimension

(B) 2-Dimensions

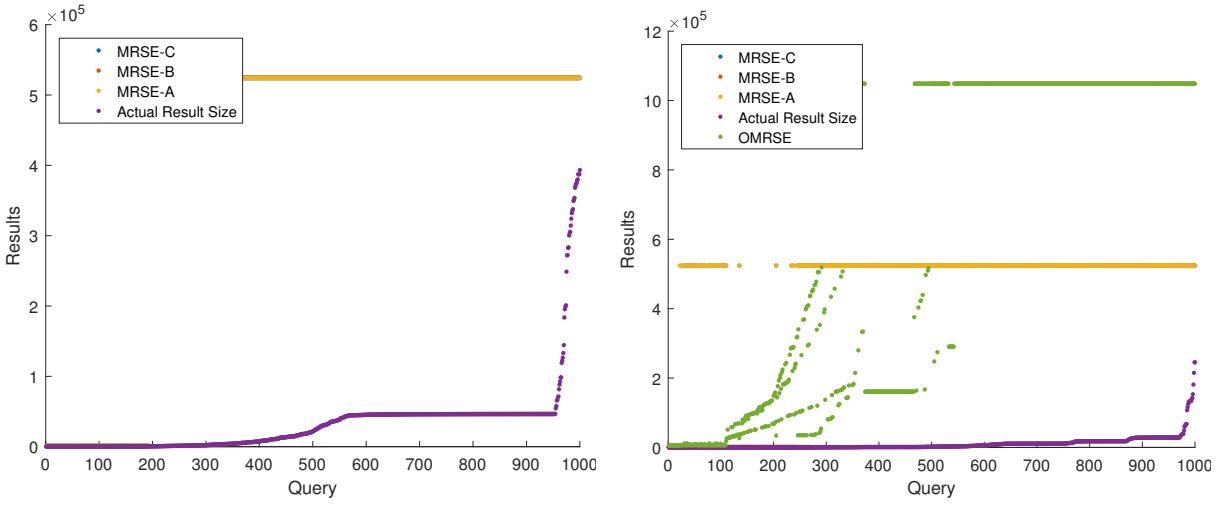


(C) 3-Dimensions



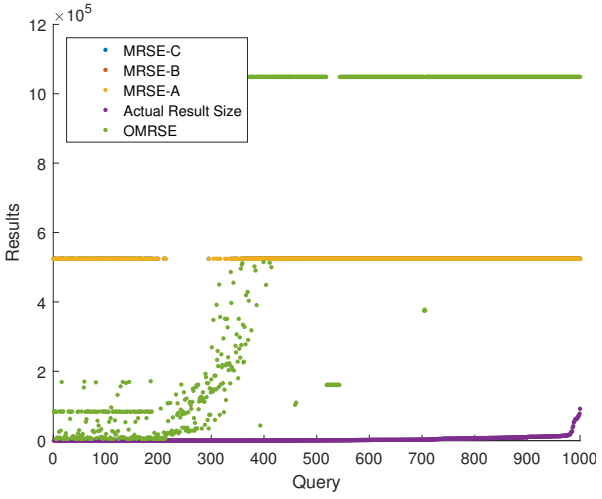
(D) 4-Dimensions

FIGURE 4.5: 20% Range Query Size for each dimension (s=2)

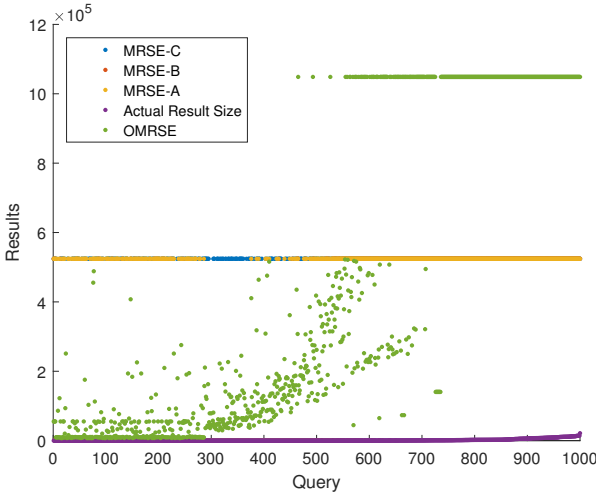


(A) 1-Dimension

(B) 2-Dimensions

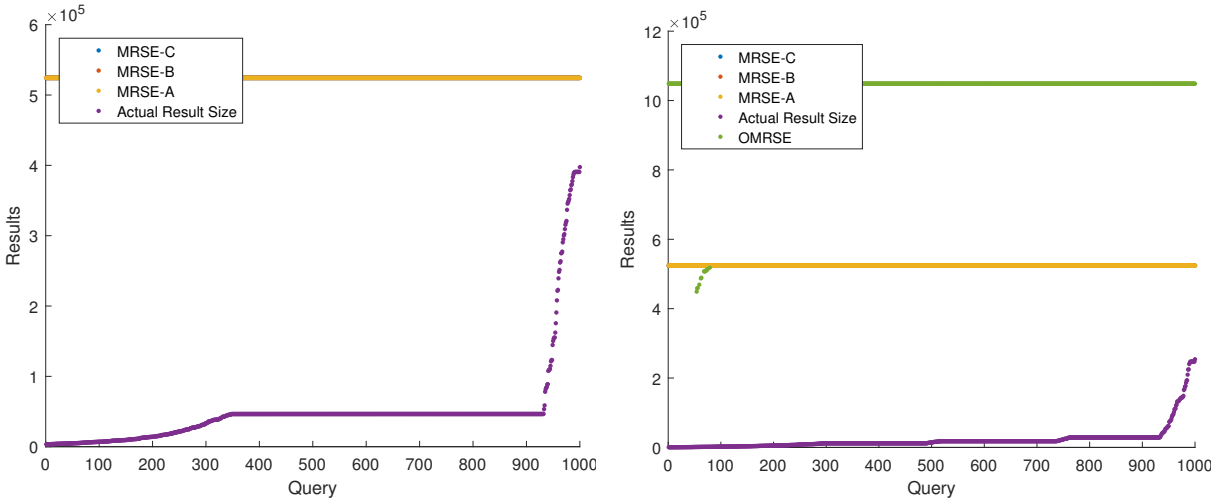


(C) 3-Dimensions



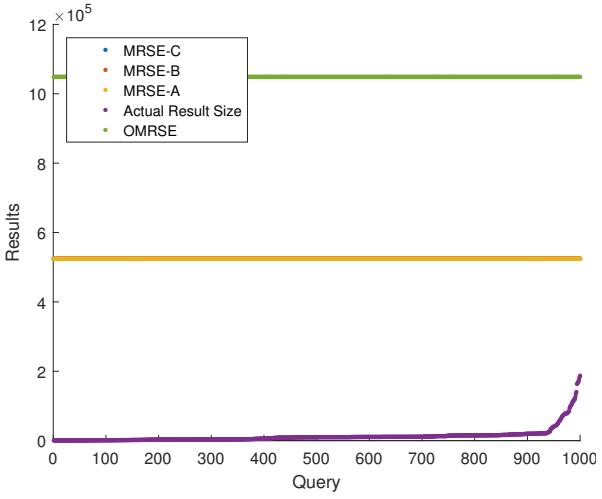
(D) 4-Dimensions

FIGURE 4.6: 40% Range Query Size for each dimension (s=2)

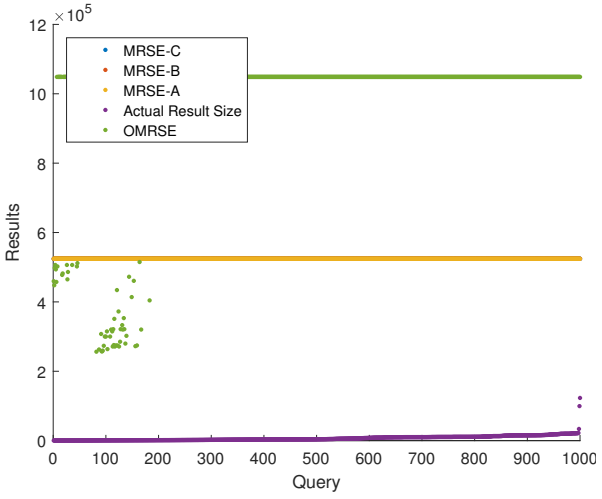


(A) 1-Dimension

(B) 2-Dimensions

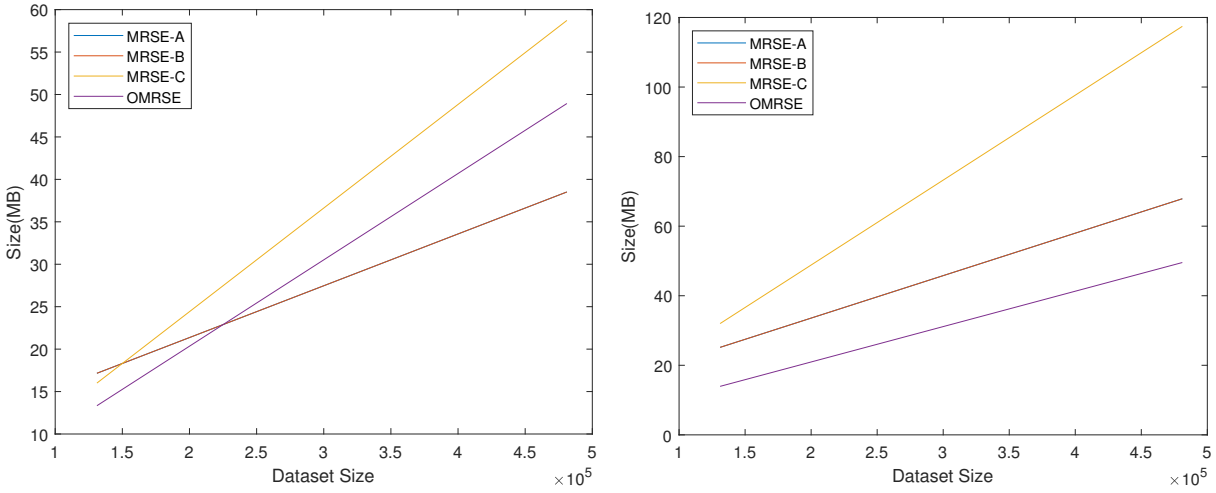


(C) 3-Dimensions



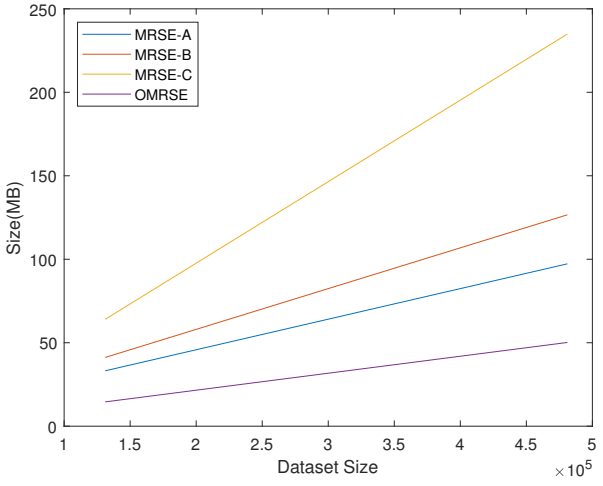
(D) 4-Dimensions

FIGURE 4.7: 60% Range Query Size for each dimension (s=2)

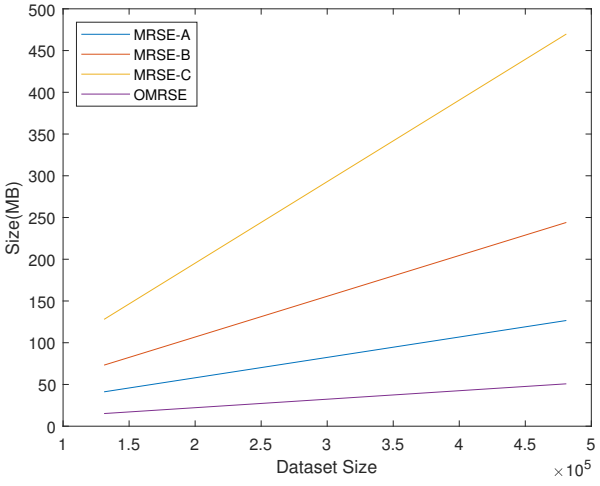


(A) 1-Dimension

(B) 2-Dimensions

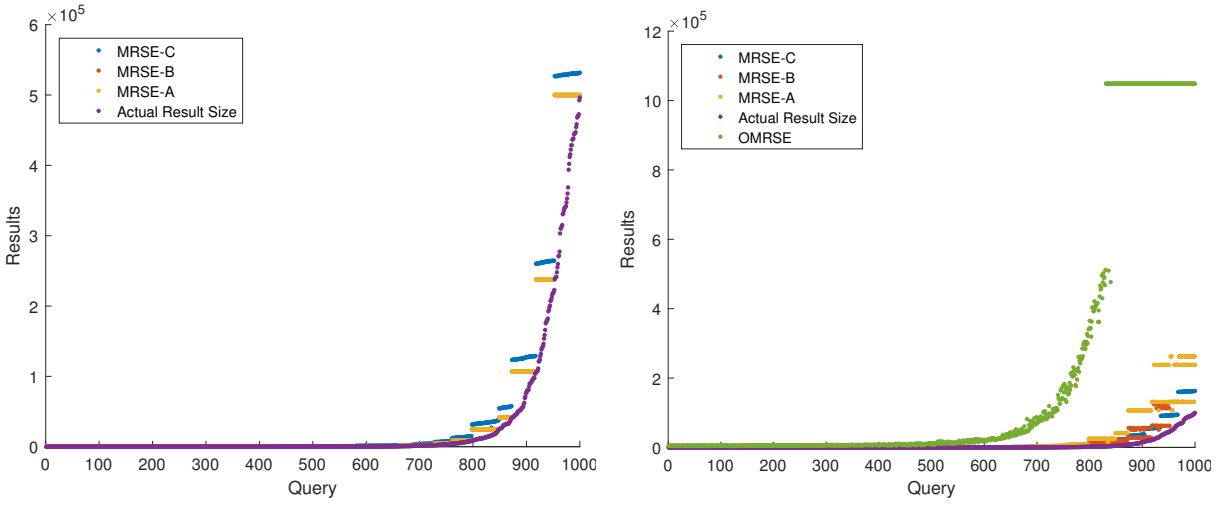


(C) 3-Dimensions



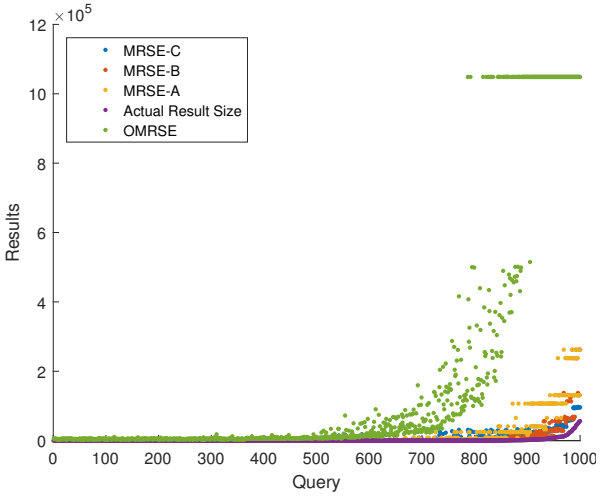
(D) 4-Dimensions

FIGURE 4.8: Index Size (s=2)

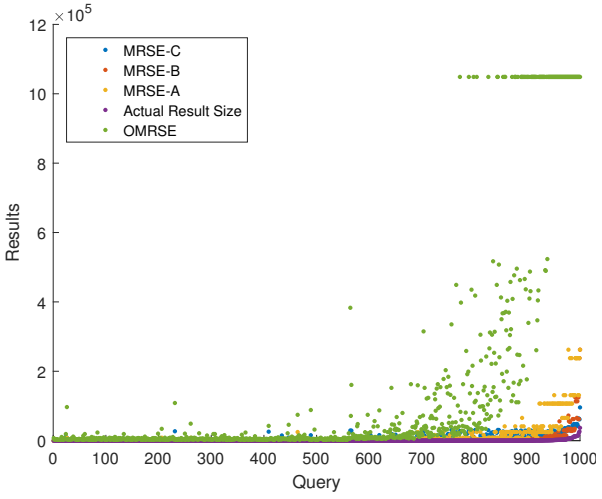


(A) 1-Dimension

(B) 2-Dimensions

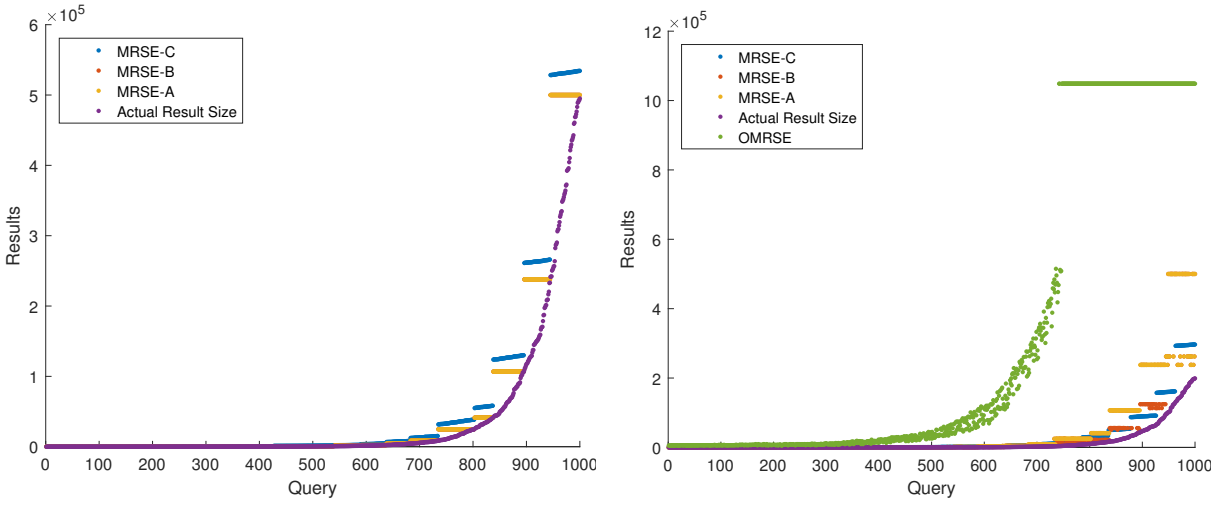


(C) 3-Dimensions



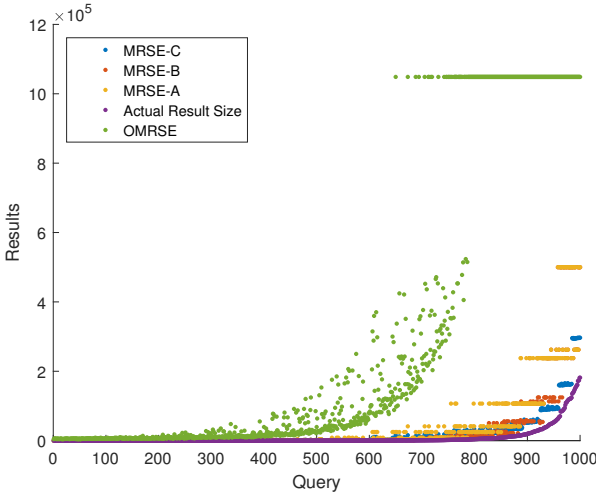
(D) 4-Dimensions

FIGURE 4.9: 20% Range Query Size for each dimension (s=ALL) (Synthetic data)

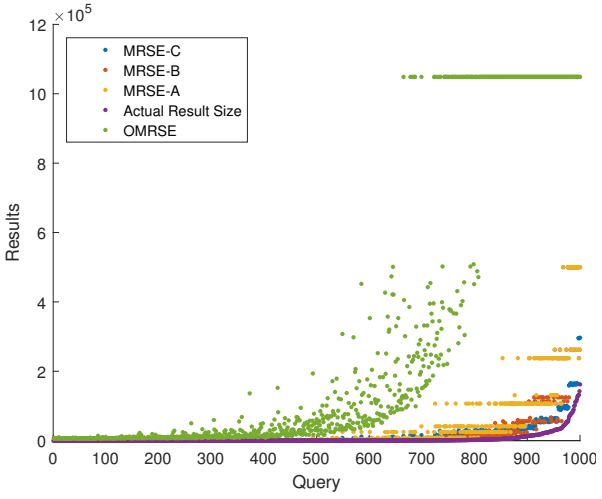


(A) 1-Dimension

(B) 2-Dimensions

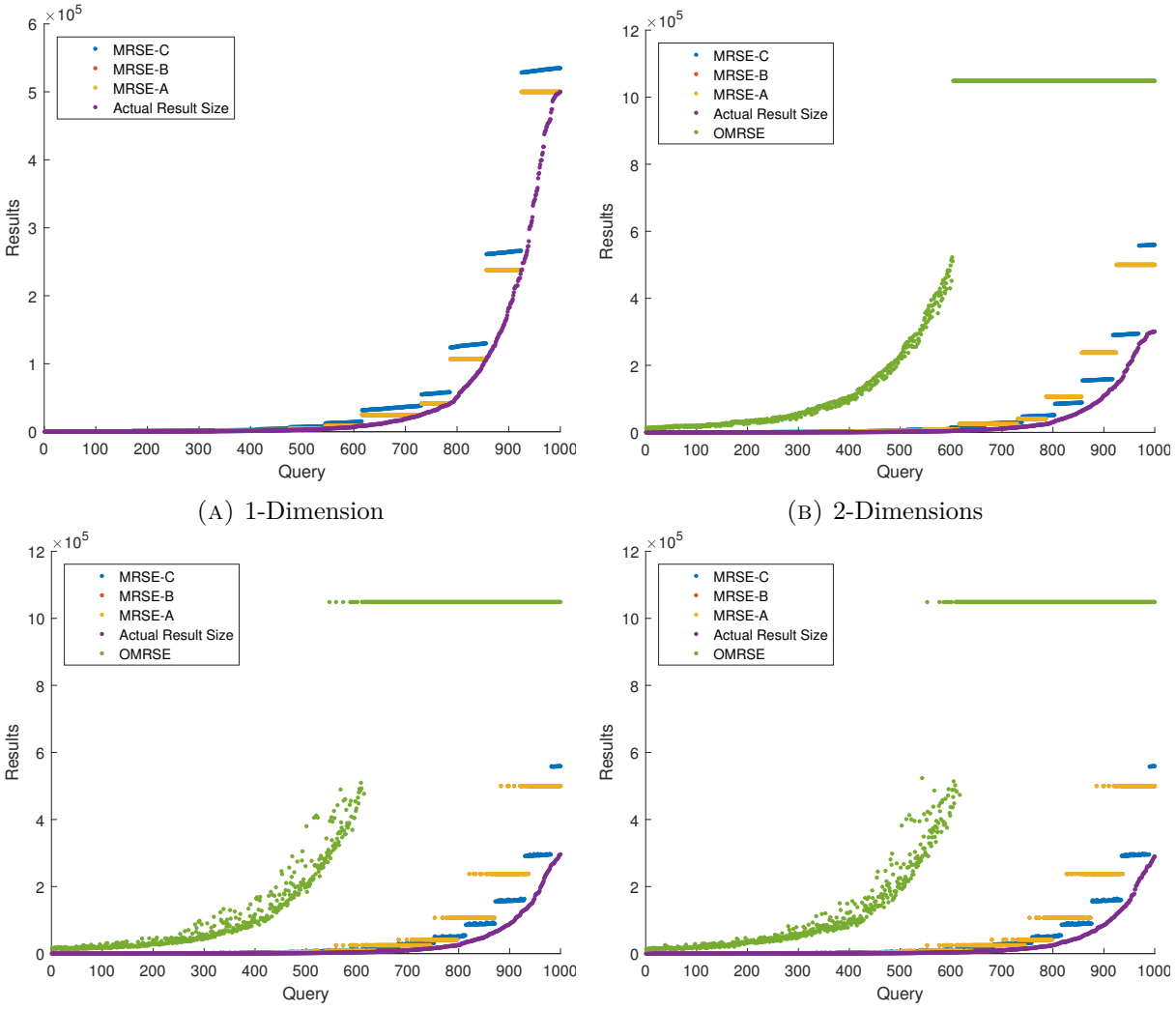


(C) 3-Dimensions



(D) 4-Dimensions

FIGURE 4.10: 40% Range Query Size for each dimension (s=ALL) (Synthetic data)



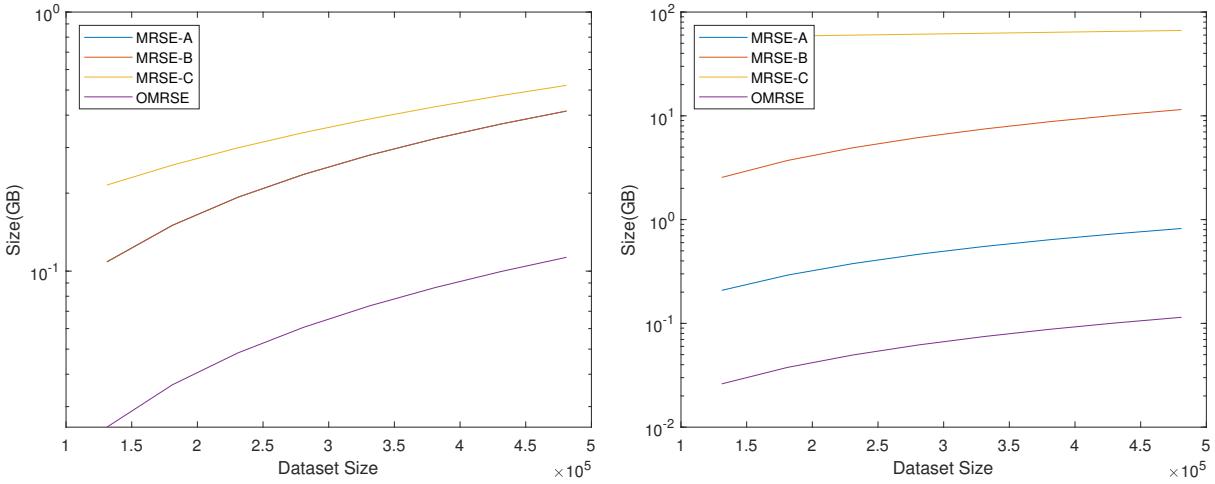
(A) 1-Dimension

(B) 2-Dimensions

(C) 3-Dimensions

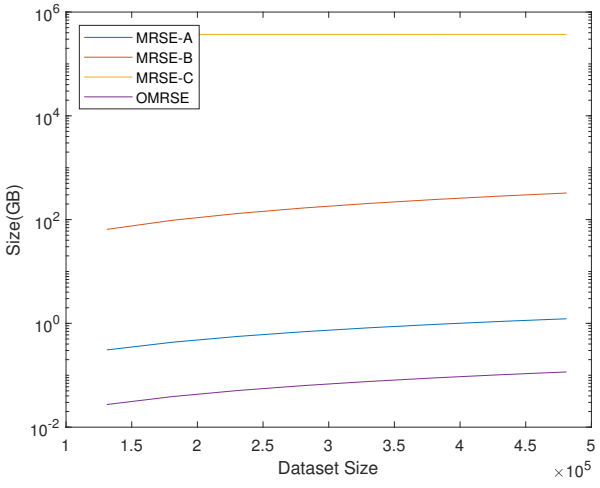
(D) 4-Dimensions

FIGURE 4.11: 60% Range Query Size for each dimension (s=ALL) (Synthetic data)

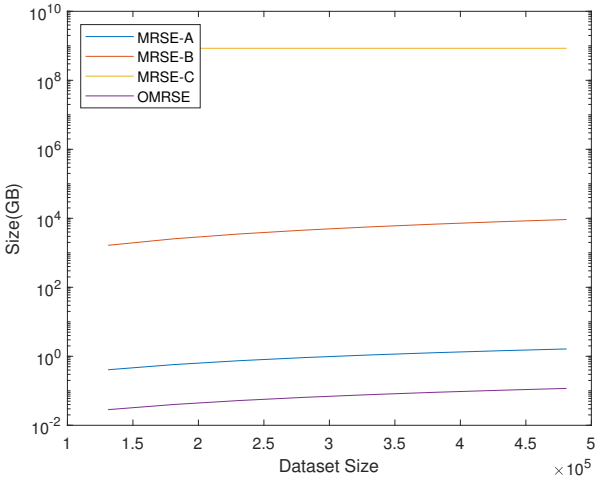


(A) 1-Dimension

(B) 2-Dimensions

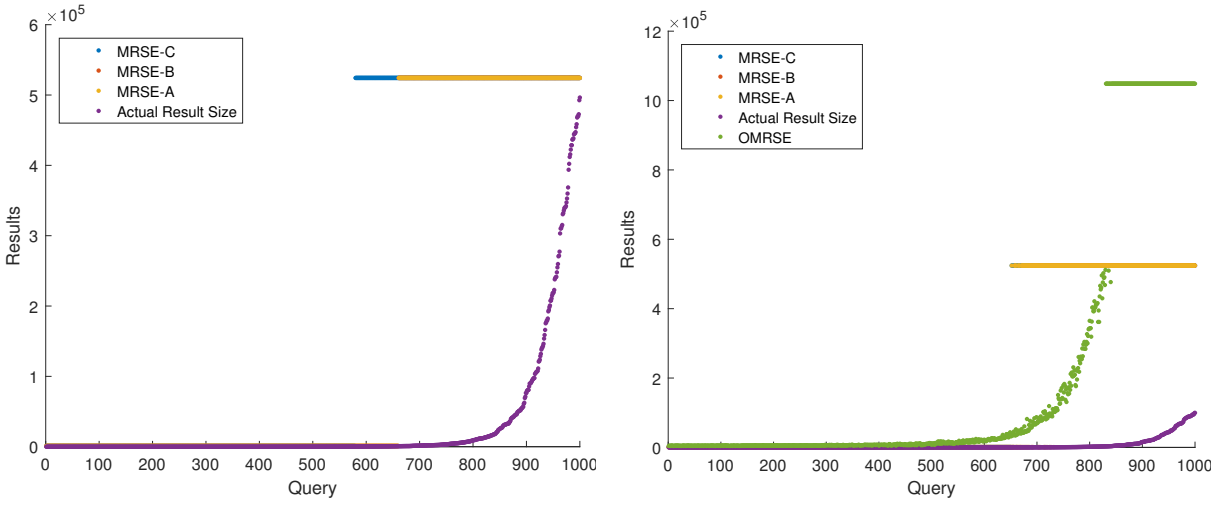


(C) 3-Dimensions



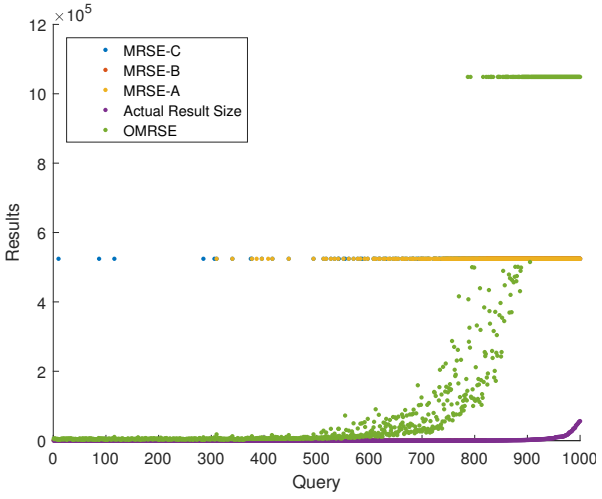
(D) 4-Dimensions

FIGURE 4.12: Index Size (s=ALL) (Synthetic data)

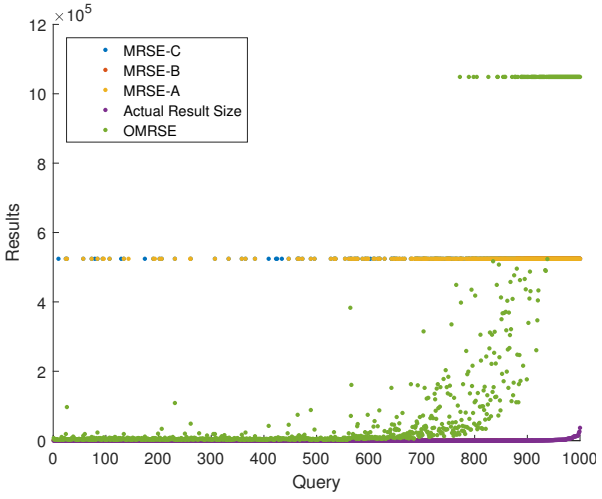


(A) 1-Dimension

(B) 2-Dimensions

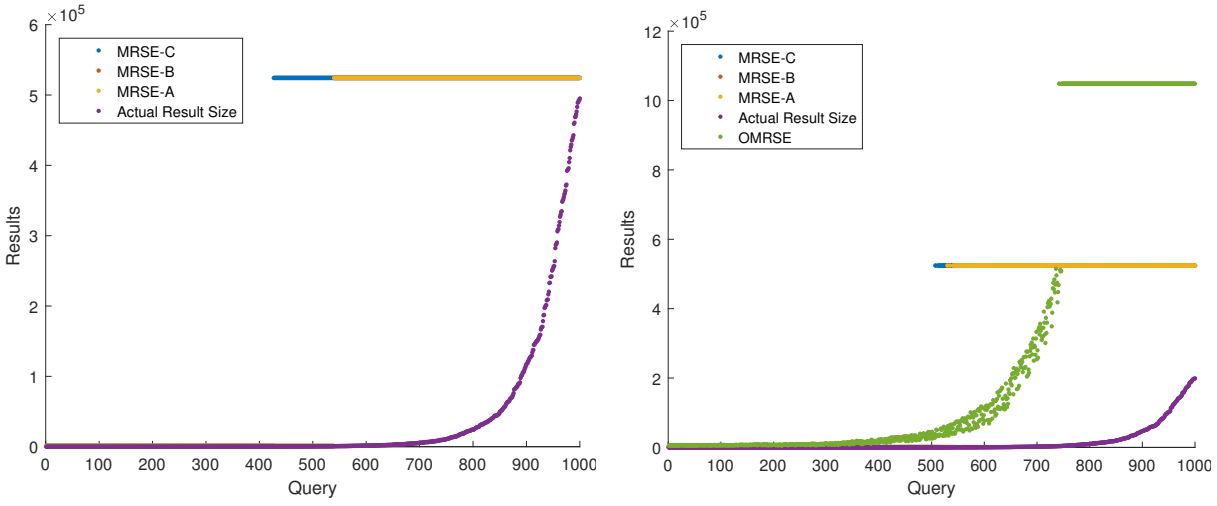


(C) 3-Dimensions



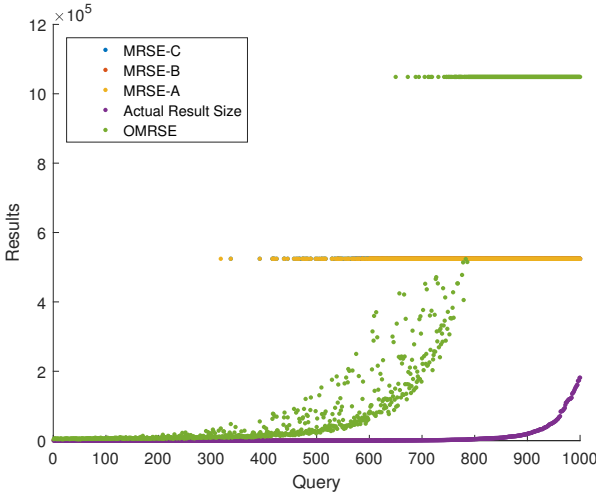
(D) 4-Dimensions

FIGURE 4.13: 20% Range Query Size for each dimension (s=2) (Synthetic data)

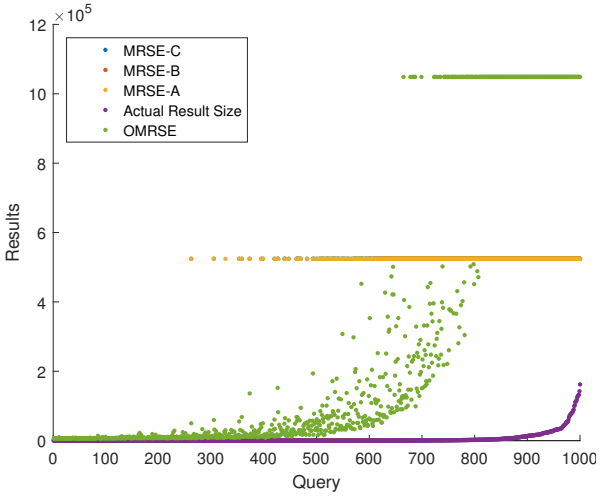


(A) 1-Dimension

(B) 2-Dimensions

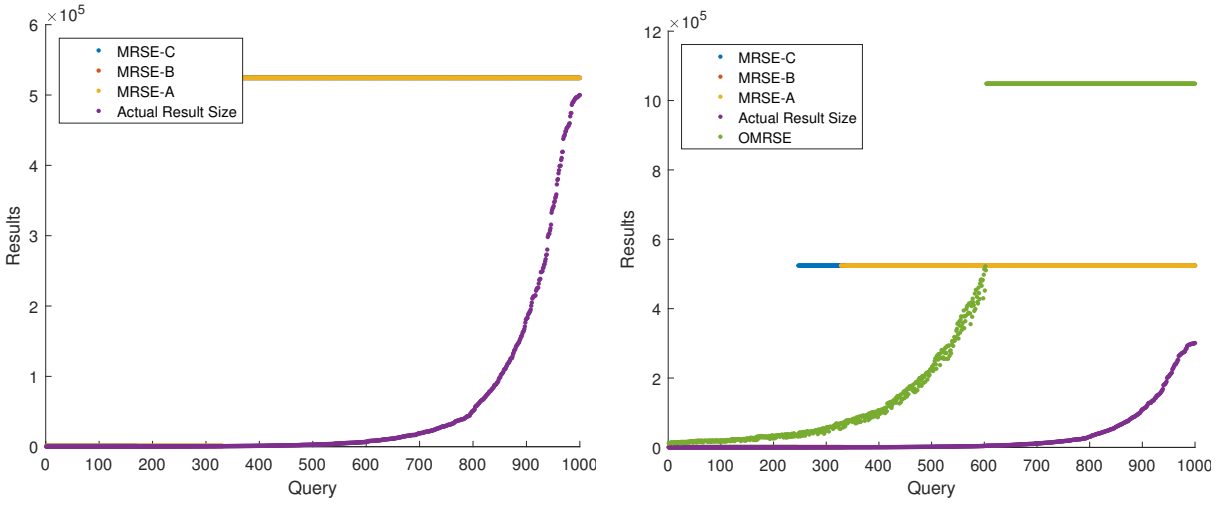


(C) 3-Dimensions



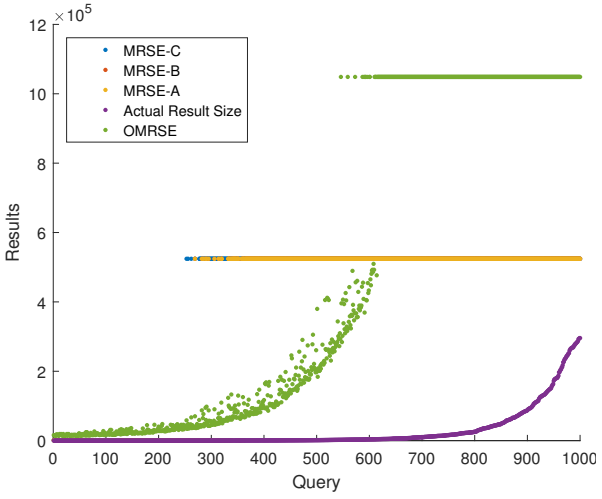
(D) 4-Dimensions

FIGURE 4.14: 40% Range Query Size for each dimension (s=2) (Synthetic data)

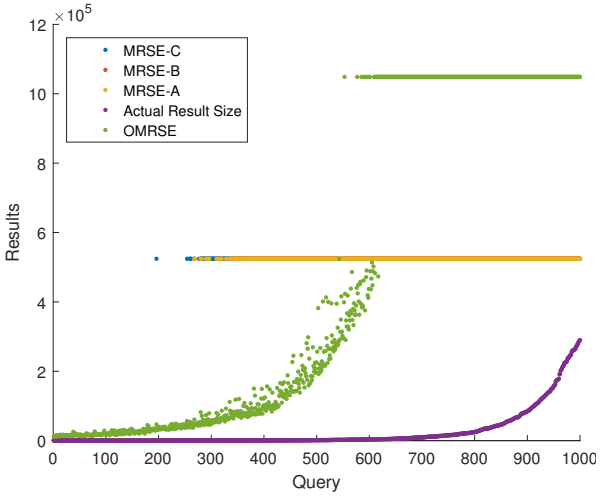


(A) 1-Dimension

(B) 2-Dimensions

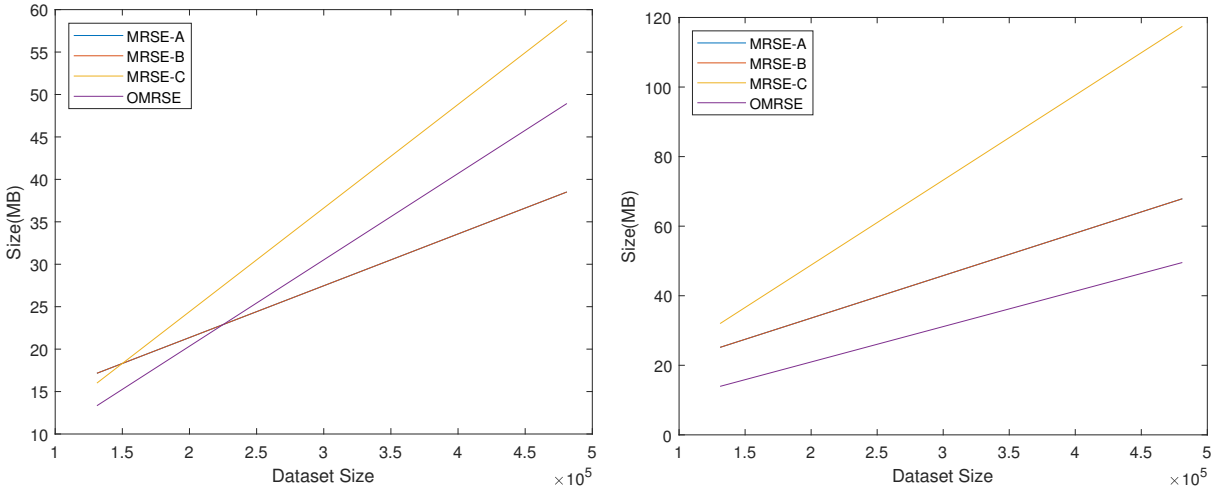


(C) 3-Dimensions



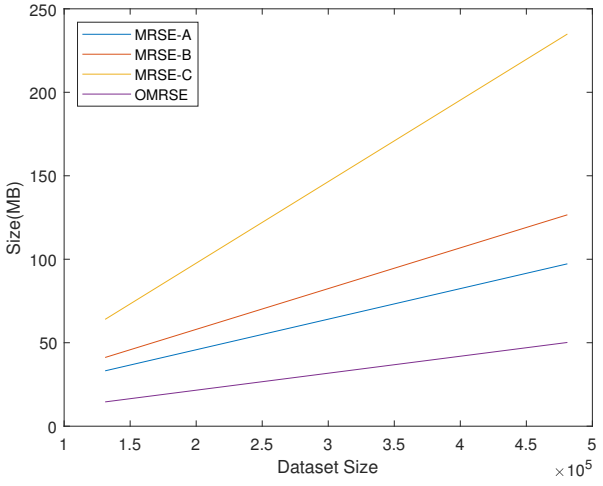
(D) 4-Dimensions

FIGURE 4.15: 60% Range Query Size for each dimension (s=2) (Synthetic data)

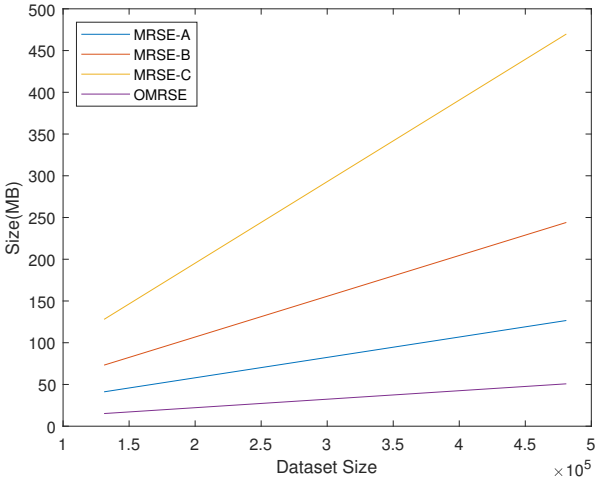


(A) 1-Dimension

(B) 2-Dimensions



(C) 3-Dimensions



(D) 4-Dimensions

FIGURE 4.16: Index Size (s=2)

Chapter 5

Conclusions and Future Work

In this work, we study the problem of Multi-Dimensional Range SE (MRSE) schemes and we provide four new novel schemes with different security and efficiency trade-offs (MRSE-A, MRSE-B, MRSE-C, OMRSE). Our MRSE-A, MRSE-B and MRSE-C constructions are based on extending prior RSE schemes to multiple dimensions, while OMRSE combines ideas from R-trees (very efficient data structures for multi-dimension range queries with linear index size) and ORAMs (schemes that are more secure and expensive than SE schemes). We experimentally show that OMRSE scheme is not only more secure than MRSE schemes but it has comparable/better search performance (up to 2× slow-down,up to 100× speed-up) than MRSE schemes when all are tuned to have similar space. In our future work, we could try to combine ORAM techniques with other spatial indexes in order to further improve the performance of MRSE/OMRSE schemes. Another importance research direction is to efficiently and securely combine MRSE/OMRSE schemes with other query operators, such as join and group-by queries.

References

[1] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *SOSP*, 2011.

[2] Stephen Tu, M Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5):289–300, 2013.

[3] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference Attacks on Property-Preserving Encrypted Databases. In *CCS*, 2015.

[4] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical Techniques for Searches on Encrypted Data. In *SP*, 2000.

[5] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *EUROCRYPT*, 2017.

[6] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *CRYPTO*, 2013.

[7] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel Rosu, and Michael Steiner. Rich Queries on Encrypted Data: Beyond Exact Matches. In *ESORICS*, 2015.

[8] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. Practical Private Range Search Revisited. In *SIGMOD*, 2016.

[9] Seny Kamara and Tarik Moataz. Sql on structurally-encrypted databases. *IACR*, 2016.

[10] Benny Pinkas and Tzachy Reinman. Oblivious RAM Revisited. In *CRYPTO*. 2010.

[11] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path Oram: An Extremely Simple Oblivious Ram Protocol. In *CCS*, 2013.

[12] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.

[13] Craig Gentry. Computing Arbitrary Functions of Encrypted Data. *Commun. of the ACM*, 2010.

[14] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, Minos Garofalakis, and Charalampos Papamanthou. Practical private range search in depth. *TODS*, 2018.

[15] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. Secure Multidimensional Range Queries over Outsourced Data. *VLDB J.*, 21(3):333–358, 2012.

[16] Peng Wang and Chinya V Ravishankar. Secure and efficient range queries on outsourced databases using rp-trees. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 314–325. IEEE, 2013.

[17] Jialin Chi, Cheng Hong, Min Zhang, and Zhenfeng Zhang. Fast multi-dimensional range queries on encrypted cloud databases. In Selçuk Candan, Lei Chen, Torben Bach Pedersen, Lijun Chang, and Wen Hua, editors, *Database Systems for Advanced Applications*, 2017.

[18] Panagiotis Karras, Artyom Nikitin, Muhammad Saad, Rudrika Bhatt, Denis Antyukhov, and Stratos Idreos. Adaptive indexing over encrypted numeric data. In *SIGMOD*, 2016.

[19] Caleb Horst, Ryo Kikuchi, and Keita Xagawa. Cryptanalysis of comparable encryption in sigmod'16. In *SIGMOD*, 2017.

[20] Jon Louis Bentley and Jerome H Friedman. Data structures for range searching. *CSUR*, 1979.

[21] Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In *CCS*, 2014.

[22] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *CCS*, 2006.

[23] Melissa Chase and Seny Kamara. Structured Encryption and Controlled Disclosure. In *ASIACRYPT*, 2010.

[24] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic Searchable Symmetric Encryption. In *CCS*, 2012.

[25] Seny Kamara and Charalampos Papamanthou. Parallel and Dynamic Searchable Symmetric Encryption. In *FC*, 2013.

[26] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical Dynamic Searchable Encryption with Small Leakage. In *NDSS*, 2014.

[27] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, M Rosu, and Michael Steiner. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *NDSS*, 2014.

[28] Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, 2006.

[29] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. *Journal of Computer Security, 2011*.

[30] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *J. ACM, 1996*, 1996.

[31] Eu-Jin Goh et al. Secure Indexes. *IACR Cryptology ePrint Archive*, 2003.

[32] Peter Van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. Computationally Efficient Searchable Symmetric Encryption. In *SDM*. 2010.

[33] Kaoru Kurosawa and Yasuhiro Ohtaki. UC-Secure Searchable Symmetric Encryption. In *FC, 2012*. 2012.

[34] Ian Miers and Payman Mohassel. IO-DSSE: Scaling Dynamic Searchable Encryption to Millions of Indexes By Improving Locality. In *NDSS*, 2017.

[35] Raphael Bost. Sofos: Forward Secure Searchable Encryption. In *CCS*, 2016.

[36] David Cash and Stefano Tessaro. The Locality of Searchable Symmetric Encryption. In *EUROCRYPT, 2014*, 2014.

[37] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. Searchable Symmetric Encryption: Optimal Locality in Linear Space via Two-Dimensional Balanced Allocations. In *STOC, 2016*, 2016.

[38] Ioannis Demertzis and Charalampos Papamanthou. Fast searchable encryption with tunable locality. In *SIGMOD*, 2017.

[39] Ioannis Demertzis, Dimitrios Papadopoulos, and Charalampos Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. *CRYPTO*, 2018.

[40] Elaine Shi, John Bethencourt, T-HH Chan, Dawn Song, and Adrian Perrig. Multi-Dimensional Range Query over Encrypted Data. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 350–364. IEEE, 2007.

[41] Yanbin Lu. Privacy-preserving logarithmic-time search on encrypted data in cloud. In *Proc. of NDSS*, 2012.

[42] Ningning Cui, Xiaochun Yang, Leixia Wang, Bin Wang, and Jianxin Li. Secure range query over encrypted data in outsourced environments. In Jian Pei, Yannis Manolopoulos, Shazia Sadiq, and Jianxin Li, editors, *Database Systems for Advanced Applications*, 2018.

[43] H. G. Do and W. K. Ng. Multidimensional range query on outsourced database with strong privacy guarantee. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, 2016.