Attacks on Encrypted Range Search Schemes in Multiple Dimensions

Francesca Falzon*
francesca_falzon@brown.edu
Brown University
University of Chicago

Evangelia Anna Markatou*
markatou@brown.edu
Brown University

Zachary Espiritu
zachary_espiritu@brown.edu
Brown University

Roberto Tamassia
roberto@tamassia.net
Brown University

ABSTRACT

We present the first systematic security evaluation of multi-attribute range search schemes on symmetrically encrypted data. We present four database reconstruction attacks that apply to a broad class of schemes and rely on volume and search pattern leakage. For schemes achieving efficiency by decomposing a query into a small number of subqueries, we further show how to exploit their structure pattern, i.e., co-occurrences of subqueries. We introduce a flexible framework for building secure range search schemes by adapting a broad class of geometric search data structures (including range trees and quadtrees) to operate on encrypted data. We give four concrete range search schemes within our framework that support queries on an arbitrary number of dimensions (attributes) and offer a sliding scale of efficiency and security trade-offs. We provide a security proof for any scheme derived from our framework and a thorough analysis of the leakage of our concrete schemes, characterizing the set of equivalent databases and demonstrating information theoretic limitations on reconstruction attacks. Our attacks are the first that do not require the observation of the access pattern to reconstruct data from range queries in two and higher dimensions. Our work shows that for range queries, structure pattern leakage can be as vulnerable to attacks as access pattern leakage. We give a comprehensive evaluation of our schemes and attacks with a complexity analysis, a prototype implementation, and an experimental assessment on real-world datasets.

KEYWORDS

Encrypted Database; Database Reconstruction; Attack

1 INTRODUCTION

With the rise of cloud services, there is a growing need for schemes that support complex privacy-preserving queries. In this paper, we study the security of schemes that support range-queries over multi-attribute data. One solution for supporting private range queries is to use theoretical primitives like fully-homomorphic encryption [19] or oblivious RAM [21]. While they offer the best security guarantees, these solutions are not yet practical. As an alternative, solutions for private range queries have been proposed using *searchable symmetric encryption* (SSE) (see, e.g., [5, 6, 8–10, 12, 20, 30–32, 43, 45, 51]). SSE schemes offer the following tradeoff: in exchange for efficiency they reveal some well-defined information, or *leakage*, about the queries and underlying data.

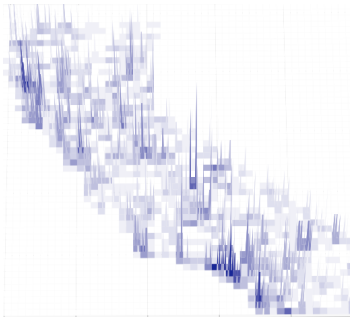


Figure 1: Reconstruction by our attack on the range tree scheme with uniform range cover (Sections 4.2 and 5.2) for the Cali dataset over a domain with 1024×1024 points. The bar heights represent the number of records at each domain point. The attack succeeds in 68s.

Leakage typically occurring in SSE schemes includes one or more of the following: ***search pattern*** (whether two query tokens refer to the same query); ***volume pattern*** (number of records in the query response); and ***access pattern*** (individually and deterministically encrypted records in query responses).

Existing efficient schemes support range queries on only single-attribute (1D) data. In this paper, we evaluate the security of a broad class of schemes that support range queries over multi-attribute data. We first give a general framework for building such schemes and a generic security proof for them. The schemes from prior work most related to those developed within our framework are by Demertzis et al. [13, 14], who present 1D range schemes with storage and security trade-offs, and by Faber et al. [16], who build on the SSE scheme in [9] to support 1D range, substring, wild-card, and phrase queries. We extend these schemes to support multi-attribute queries, analyze their leakage, and present attacks on them. Our work is the first to systematically analyze the security of schemes for encrypted range search on more than two attributes. We show that volume and search pattern—when combined with the structural information of the underlying range search data structure—is as detrimental to security as access and search pattern. One of our attacks works on a wide class of range encrypted multimap schemes that achieve efficiency allowing for false positives in responses, and are regarded to be amongst the most secure. We evaluate our schemes and attacks using real-world datasets.

Several SSE schemes have been developed for single-attribute (1D) range queries on encrypted databases (see, e.g., [3, 27, 29, 49, 54]). Prior work on multi-attribute range query schemes uses other

*Both authors contributed equally to this research.

Table 1: Comparison of our schemes for range queries over an arbitrary number of attributes of an encrypted database, and of our reconstruction attacks on the schemes. Regarding scheme complexity, we show the query size, response size, and server space. The query time at the client and server is proportional to query size plus response size. The client space is $O(1)$ but for up to polylog temporary space when a query is issued. For each scheme, we also list selected related schemes and attacks from previous work, which are limited to 1D and 2D. Regarding attack complexity, we exclude the time to read the leakage, which is the same for any attack and depends on the database size and the domain size, and the required number of queries reported assumes a uniform query distribution. Notation: range size R, result size r, database size n, domain size m, number of query attributes (dimension of the domain) d.

Scheme	Selected Related Work		Scheme Complexity			Attack Complexity			
	Schemes (all 1D)	Attacks	Query size	Resp. size	Space	Reconstr. space	Runtime	Space	# Queries
Linear	Naive [13]	1D [25, 33], 2D [17]	R	r	$m+n$	$2^d\,(d!)$	m^5	m^3	$m^{2-\frac{1}{d}}\log^2 m$
Range URC	Range [13, 16]	–	$\log^d R$	r	$m+n\log^d m$	$2^d\,(d!)$	$m^2\log^d m$	$m^2\log^d m$	$m^2\log m$
Range BRC	Range [13]	–	$\log^d R$	r	$m+n\log^d m$	$2^d\,(d!)$	m^4	$m^2\log^d m$	$m^2\log m$
QDAG SRC	TDAG [13]	1D [37]	1	$r+R^d$	$m+n\log m$	$\geq 2^{d+2^{(d-1)}}\,(d)\,(d!)$	–	–	$m^4\log m$

frameworks. Shi et al. [50] and Wang et al. [53] use public-key cryptography, whereas Kermanshahi et al. [34] use homomorphic encryption to support multi-attribute range queries. De Capitani di Vimercati et al. [15] index multi-dimensional encrypted data by recursively partitioning records into boxes, thus taking steps toward a general scheme, but do not provide a formal leakage analysis.

Leakage analysis of SSE schemes has been studied in a passive adversarial setting e.g. [2, 7, 28, 44, 47]. Recent attention has been devoted to exploiting the interplay between volume and search pattern e.g. Oya and Kerschbaum [44] and Blackstone et al. [2]. Database reconstruction attacks have been presented against schemes supporting 1D range queries. The first such attack by Kellaris et al. [33] was followed by more efficient attacks for 1D queries using access (e.g. [25, 36, 38, 41]) and volume pattern (e.g. [24, 26, 37]).

Two attack works most related to our Linear attack are the generic 2D database reconstruction attacks in [17, 40]. Unlike these works, we give attacks on concrete range schemes and our attacks work on databases of *two and higher* dimensions. The closest prior attack to our SRC attack is by Kornaropoulos, Papamanthou, and Tamassia [37], who attack 1D response hiding schemes. In [14], Demertzis et al. note their schemes are susceptible to attacks but do not give a full description or analysis.

Our contributions are summarized as follows:

- We present the ***first attacks on encrypted database schemes that support range queries on an arbitrary number of attributes in a standard scenario where the scheme leaks search pattern and volume pattern***. Previous attacks were either limited to 1D queries, or 2D queries in the scenario where the attacker additionally observes the access pattern. (Section 5)
- We give ***attacks on efficient schemes that decompose a query into a small (polylog) number of subqueries.*** These schemes also leak what we call ***structure pattern***, i.e., the pattern of co-occurrence of subqueries. ***Structure pattern leakage is inherent in schemes derived from standard multidimensional search data structures*** (e.g., the 1D logarithmic scheme in [13], which is derived from the range tree). (Sections 5.1 and 5.2)
- We show ***attacks on schemes that achieve efficiency without leaking the structure pattern by allowing responses to contain false positives***. Such schemes are considered the gold standard, since recent attacks have been limited to only 1D queries [37]. (Section 5.3)
- We introduce a ***general framework for building range search schemes for encrypted databases*** by adapting a broad class of geometric search data structures (e.g. range trees and quadtrees) to operate on encrypted data. We provide a ***generic security proof for any scheme derived from our framework***. (Section 3)
- We ***present four concrete schemes derived from the framework and analyze their complexity***. (Section 4)
- We ***characterize the set of equivalent databases under the leakage of our schemes*** and demonstrate information theoretic limitations on the reconstruction ability of a passive adversary. (Section 5)
- We evaluate our schemes and our attacks with a ***theoretical complexity analysis, prototype implementation, and experimental evaluation on real-world datasets***. (Section 6)

Table 1 compares our concrete schemes for multi-attribute range queries on an encrypted database and our reconstruction attacks on them. Our work demonstrates the pitfalls of extending 1D encrypted range search schemes to higher dimensions and helps inform future research on expressive queries on multidimensional data.

2 PRELIMINARIES

Given integers a, b with $a \leq b$, let $[a] = \{1, 2, \ldots, a\}$ and let $[a, b] = \{a, a+1, \ldots, b\}$. Let $m_1, \ldots, m_d$ be positive integers and $d \geq 2$. A *d-attribute database*, or a *d-dimensional database*, D is an injective mapping from a domain $\mathcal{D} = [m_1] \times \cdots \times [m_d]$ to a set of records of $O(1)$ size. We denote the set of records with domain value $x = (x_1, \ldots, x_d) \in \mathcal{D}$ as $D[x]$. A *d-dimensional range query* is a hyper-rectangle $[a_1, b_1] \times \cdots \times [a_d, b_d]$ where $[a_i, b_i] \subseteq [1, m_i]$ denotes the range in the i-th dimension.

We say that points p and p' are *neighbors* if they share every coordinate but one, and in the remaining coordinate, their values differ by one. We call a set of contiguous points of D that only differ in the same single coordinate a *one-dimensional section*.

We use double-brace notation to denote a multiset, e.g. {{1,1,4,5,7}}.

Symmetric Encryption. A *symmetric encryption scheme* is a tuple of polynomial-time algorithms $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows. Gen is a probabilistic algorithm that takes as input a security parameter λ and outputs a secret key K. Enc is a probabilistic algorithm that takes as input a key K and a message m and returns a ciphertext c. Dec is a deterministic algorithm that takes as input a key K and a ciphertext c and returns a message M. For correctness we require that $\mathsf{Dec}(K, \mathsf{Enc}(K, M)) = M$ for all K and M. We further require that the scheme is CPA-secure.

Appendix. Additional material, including proofs of theorems and lemmas, can be found in the Appendix.

2.1 EMM Definition and Security Model

EMM scheme syntax. Our range search schemes on encrypted data are built using an *encrypted multimap* (EMM) scheme in a generic manner. A *multimap* is a map that takes labels from a label space $\mathbb{L}$ to sets of values from a value space $\mathbb{V}$ i.e. $\mathsf{MM} : \mathbb{L} \mapsto 2^{\mathbb{V}} \cup \{\perp\}$ where $\perp$ indicates an uninitialized value. Given a multimap MM, we denote the set of values associated to label ℓ as $\mathsf{MM}[\ell]$.

Definition 1 ([10]). *An **encrypted multimap scheme** is a tuple of algorithms* $\Sigma = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Eval}, \mathsf{Result})$*, where*

- $\Sigma.\mathsf{Setup}$*: (probabilistic) takes a security parameter and a multimap, and returns a secret key and an encrypted multimap.*
- $\Sigma.\mathsf{Query}$*: takes a key and label, and returns a search token.*
- $\Sigma.\mathsf{Eval}$*: takes an encrypted multimap and a search token, and returns a ciphertext.*
- $\Sigma.\mathsf{Result}$*: takes a key and a ciphertext, and returns a set of values.*

Throughout this paper, our label space $\mathbb{L}$ is the set of possible ranges over the desired domain, and the value space $\mathbb{V}$ is the set of possible record values, i.e. $\{0,1\}^*$.

EMM security model. The security of structured encryption is traditionally proven using the real-ideal paradigm [10]. The definition of adaptive security for an EMM scheme Σ is parameterized by a leakage function $\mathcal{L}^{\Sigma} = (\mathcal{L}^{\Sigma}_{\mathsf{S}}, \mathcal{L}^{\Sigma}_{\mathsf{Q}})$ which describes the exact information that a passive adversary may learn about the underlying database (formally described in Appendix A). In particular, $\mathcal{L}^{\Sigma}_{\mathsf{S}}$ captures the leakage at setup and $\mathcal{L}^{\Sigma}_{\mathsf{Q}}$ captures the leakage when a sequence of queries is issued. Using this security framework, we refer to an adaptively $(\mathcal{L}^{\Sigma}_{\mathsf{S}}, \mathcal{L}^{\Sigma}_{\mathsf{Q}})$-secure EMM scheme. The goal is to prove that the EMM scheme is indistinguishable from an ideal setting in which an algorithm simulates the response of the setup and query algorithms using only the leakage. Adaptive security of an EMM scheme is formally defined in Definition 7 in Appendix B.

2.2 REMM Definition and Security Model

Definition 2. *A **range encrypted multimap scheme** is a tuple of four algorithms* $\mathsf{REMM} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Eval}, \mathsf{Result})$*. The syntax of the algorithms is defined as those in Definition 1 with the following two changes:*

- $\mathsf{REMM.Setup}$ *takes as input a security parameter* 1^{λ} *and a multi-attribute database* D *and outputs a key* K *and an encrypted database* EMM*.*
- $\mathsf{REMM.Query}$ *takes as a input a key* K *and a range query* q *on the domain of* D *and outputs a token* t*.*

In order to support range queries, we take the label space of the underlying multimap to be the set of all possible range queries and the value space to be the set of all possible records.

For correctness we require that for all d-dimensional databases D with domain $\mathcal{D}$, all d-dimensional range queries q over $\mathcal{D}$, and all security parameters 1^{λ}, we have $\{D[x] : x \in q\} \subseteq V$, where $(K, \mathsf{EMM}) \leftarrow \mathsf{REMM.Setup}(1^{\lambda}, D), t \leftarrow \mathsf{REMM.Query}(K, q), C \leftarrow \mathsf{REMM.Eval}(\mathsf{EMM}, t)$, and $V \leftarrow \mathsf{REMM.Eval}(K, C)$.

REMM security model. We extend the security model in [13] to encrypted multidimensional range schemes with games $\mathbf{Real}^{\mathsf{REMM}}_{\mathcal{A}}$ and $\mathbf{Ideal}^{\mathsf{REMM}}_{\mathcal{A},\mathcal{S}}$. They are identical to the game in Definition 7 in Appendix B except that in step (1) the adversary selects a multi-attribute database D on domain D, in step (2) the adversary selects a polynomial number of range queries on the domain of D, and Σ is replaced by an encrypted multi-dimensional range scheme REMM. The adaptive security of REMM schemes is defined analogously to Definition 7.

3 GENERIC FRAMEWORK

We now present a framework for building range encrypted multimap schemes from data structures for multidimensional range search based on a search DAG. This framework generalizes many 1D range schemes and captures commonly used data-structures for range search such as range-trees, kd-trees, and quad-trees. These schemes provide a variety of trade-offs with respect to query size, response size, storage, and security. We introduce the family of range-supporting data structures and explain how to build an encrypted index from a data structure in this family. The strength and genericness of our attacks in Section 5 are a consequence of our ability to characterize entire classes of schemes via this framework.

Definition 3. *A **range-supporting data structure** for a database* D *with domain* $\mathcal{D}$ *is a pair* (G, RC)*, where:*

(i) G *is a connected directed acyclic graph (DAG).*
(ii) Each vertex v *of* G *corresponds to a range on domain* $\mathcal{D}$*, which we denote as* $v.range$ *and refer to as a **canonical range**.*
(iii) G *has a single source vertex* s *whose range is the entire domain, i.e.,* $s.range = \mathcal{D}$*. For each non-sink (non-leaf) vertex* v *of* G*, we have* $v.range = \bigcup_{(v,w)\in G} w.range$*.*
(iv) RC*, called **range covering algorithm**, is a polynomial-time algorithm that takes as input DAG* G *and a range query* q *on domain* $\mathcal{D}$*, and returns a subset* W *of vertices of* G*, called a **cover** of range* q*, such that the union of the canonical ranges of* W *includes range* q*, i.e.,* $q \subseteq \bigcup_{w\in W} w.range$*.*

A range-supporting data structure can be used to perform range queries by precomputing and storing the responses to all the canonical ranges of the scheme. To perform a range query q, we use the range covering function to find a cover W of q, retrieve the responses to the canonical queries for the nodes of W, and return their union as the response to q.

The response to a range query q may have ***false positives***, i.e., points of of the database outside of range q, which will have to be filtered out to obtain the exact response. To avoid false positives, we use a data structure where the cover W returned by the range covering function is such that the union of the canonical ranges of W is equal to range q, i.e., $q = \bigcup_{v\in W} v.range$.

The theorem below gives a necessary condition for a range-supporting data structure to be without false positives.

Theorem 1. *Let* (G, RC) *be a range-supporting data structure for a domain* $\mathcal{D}$ *such that the answer to any range query has no false positives. Then, for every domain point* $x \in \mathcal{D}$*, there is a node* v *of* G *with canonical range* $v.range = x$*.*

Note that, for any DAG, there is a trivial range covering algorithm that returns the source node of the DAG for every query,

Algorithm 1: $\mathsf{BRC}(T, q, v)$

```
// Invoked with BRC(T, q, s), where s is the root (source) of T
Label v as explored
W ← ∅
if v.range ⊆ q then
  W ← {v}
else
  if v.range ∩ q ≠ ∅ then
    for (v, w) ∈ T and w is not labeled as explored do
      W ← W ∪ BRC(T, q, w)
return W
```

Algorithm 2: $\mathsf{SRC}(G, q, v)$

```
// Invoked with SRC(G, q, s), where s is the source of G
Label v as explored
cand ← null
if q ⊆ v.range then
  cand ← v
  for (v, w) ∈ G and w is not labeled as explored do
    {t} ← SRC(G, q, w)
    if |t.range| < |cand.range| then
      cand ← t
return {cand}
```

i.e., the trivial cover consisting of the entire domain. In general, this algorithm will cause false positives. To avoid false positives one needs to use a cover with multiple nodes. When the DAG is a tree, T, whose leaves are associated with the domain points (Theorem 1) and for each internal node v, the canonical ranges of the children of v are a partition of $v.range$, Algorithm 1, called ***best range cover*** (BRC), produces a cover of the query range with the minimum number of nodes. This algorithm generalizes the classic one-dimensional range tree scheme to an arbitrary partition of the domain.

Theorem 2. *Let (T, BRC) be a range-supporting data structure whose DAG is a tree T such that*

(1) the canonical range of the leaves (sinks) of T are in 1-1 correspondence with the domain points $x \in \mathcal{D}$;
(2) for each internal node v of T, the canonical ranges of the children (successors) of v are a non-trivial partition of the canonical range of v, i.e., $v.range = \bigcup_{(v,w)\in T} w.range$ and $\bigcap_{(v,w)\in T} w.range = \emptyset$.

Then for any range query q, the cover returned by BRC has no false positives and is unique and minimal (i.e. smallest number of nodes).

Corollary 1. *Let (T, BRC) be a range-supporting data structure whose DAG is a tree T such that only (2) holds. For any range query q that is the union of canonical ranges of leaves, the cover returned by BRC has no false positives and is the unique minimal cover.*

When false positives are acceptable in the query answer, it may be desirable to have a cover consisting of a single node, which is accomplished by Algorithm 2, called ***single range cover*** (SRC).

Theorem 3. *Let (G, SRC) be a range-supporting data structure. For any range query q, the cover v returned by SRC minimizes the number of domain points of the cover outside of q, i.e. the number of potential false positives.*

Since many different domain-dependent data structures can be encoded as a DAG with the properties of Definition 3, we now describe a generic scheme that supports range queries given any range-supporting data structure.

Definition 4. *Given security parameter λ, an encrypted multimap scheme Σ, and a range-supporting data structure (G, RC) for a database D over domain $\mathcal{D}$, the **generic range encrypted multimap scheme scheme** $\mathsf{GenericRS}(1^\lambda, D, \mathcal{D}, \Sigma, (G, \mathsf{RC}))$ is derived as follows. The client initializes a multimap MM, and for every node v of G sets $\mathsf{MM}[v.range] \leftarrow \{D[x] : x \in v.range\}$. Thus, multimap MM associates each canonical range with the set of database records contained in the range. i.e., with the response to the query for that range. The client then creates from MM an encrypted multimap EMM with secret key K using EMM scheme Σ, and outsources EMM to the server. To perform a range query q, the client computes $\mathsf{RC}(G, q)$ to obtain a cover W of q. Next, for each node $w \in W$, it computes a search token t_w for EMM by setting $t_w \leftarrow \Sigma.\mathsf{Query}(K, w.range)$. We refer to this set of search tokens of encrypted multimap MM associated with range query q as the **tokenset** $\mathbf{t}$ of q. The client sends tokenset $\mathbf{t}$ to the server, who then retrieves from MM the corresponding encrypted sets of records $C(t) \leftarrow \Sigma.\mathsf{Eval}(t, \mathsf{EMM})$ for $t \in \mathbf{t}$ (i.e., the encrypted responses for the canonical ranges) and returns them to the client. Finally, the client decrypts each such set $C(t)$ with $\Sigma.\mathsf{Result}(K, C(t))$.*

We give pseudocode for our generic range encrypted multimap scheme, GenericRS, in Figure 11 in Appendix C.1.

Security. Our generic range encrypted multimap scheme built from a range-supporting data structure leaks the size of the domain and the total size of the entries stored in the EMM. For each query, it leaks the tokenset and the sizes of the partial responses of each token. Thus, this scheme gives rise to an additional leakage resulting from the chosen DAG and range covering scheme, and which is parameterized by the leakage of the underlying EMM scheme. We call this leakage the ***structure pattern***.

Definition 5. *Let (G, RC) be a range-supporting data structure for a d-dimensional database D with domain $\mathcal{D}$, let MM be the multimap resulting from $\mathsf{GenericRS}$, and let Σ be the encrypted multimap scheme. Let s be the source node of G. The **structure pattern** of a range query q is*

$$\mathsf{Str}(D, q) \mapsto (\mathcal{L}_{\mathsf{Q}}^{\Sigma}(\mathsf{MM}, v.range))_{v \in \mathsf{RC}(G,q)}.$$

The leakage of GenericRS is formally characterized in the following theorem.

Theorem 4. *Given an adaptively secure EMM scheme Σ that leaks search pattern and volume pattern, and a range-supporting data structure (G, RC) for a database D with domain $\mathcal{D}$ of size m, the generic range encrypted multimap scheme $\mathsf{GenericRS}$ built from Σ and (G, RC) is adaptively $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$-secure, where:*

- $\mathcal{L}_{\mathsf{S}}(D, \mathcal{D}) = \mathcal{L}_{\mathsf{S}}^{\Sigma}(\mathsf{MM})$
- $\mathcal{L}_{\mathsf{Q}}(D, q^{(1)}, \ldots, q^{(t)}) = (\mathsf{Str}(D, q^{(i)}))_{i \in [t]}$

Note that the leakage of the scheme is heavily dependent on the DAG and range covering algorithm used. We give concrete instances of the scheme in Section 4, and we thoroughly analyze their leakage and describe attacks on them in Section 5.

4 SCHEMES

We now give concrete examples of schemes that fit the general framework; these schemes include generalizations of schemes that were presented by Demertzis et al. [13, 14] and Faber et al. [16], as well as a new scheme based a the quadtree data structure [18]. Our cryptanalysis in Section 5 is carried on entire classes of schemes captured by our framework; these classes include the prior 1D schemes [13, 14, 16].

For concreteness, we present four schemes for range search on encrypted databases. All the schemes are instances of range encrypted multimap schemes based on range-supporting data structures presented in Section 3. The schemes support multidimensional range searches on an arbitrary fixed number of dimensions (attributes), d. These schemes are derived from classic data structures for geometric searching and offer trade-offs for efficiency and security. For each scheme, we describe the associated DAG and range covering algorithm, and evaluate its storage and communication complexity. The leakage is subsequently discussed in Section 5.

Following the notation used throughout the paper, we denote the database with D and the domain with $\mathcal{D}$. We denote their sizes as $n = |D|$ and $m = |\mathcal{D}|$. The number of domain points in a query range is referred to as ***range size*** and denoted with R. The number database of records within a query range is referred to as ***result size*** and denoted with r. In our schemes, a query is issued by the client to the server as a tokenset. We refer to the number of search tokens in the tokenset as the ***query size***. A response is returned by the server to the client as a collection of encrypted sets of records (one set per search token), whose total number of records is referred to as ***response size***. Note that the response size is equal to the result size plus the number of false positives returned.

Additional information about the schemes can be found in Appendix D and a summary of their complexity in Table 1.

4.1 Linear Scheme

We first present a simple scheme, called ***linear scheme***, that offers the smallest storage at the expense of the least security.

The linear scheme indexes each record by its location. Its DAG, G_L, is a star comprising a source s adjacent to m sinks. We have that $s.range = \mathcal{D}$ and each sink v is associated with a distinct domain point $x \in \mathcal{D}$ such that $v.range = x$. For this scheme, the generic BRC algorithm takes $O(m)$ time. For better efficiency, we use the linear range covering algorithm, Algorithm 9 (LRC) in Appendix D.1.1, where in a preprocessing step, the sinks of G_L are stored in a d-dimensional array, $V[\mathcal{D}]$, indexed by the associated domain point. Details on the linear scheme are given in Appendix D.1.

4.2 Range tree Scheme

In an effort to decrease the bandwidth of the linear scheme, we present the ***range tree scheme*** based on a classic data structure [1].

A *range tree* G_{RT} on a d-dimensional domain is a recursively defined tree. Start with a binary search tree on $[m_1]$. Each node v in this tree is associated with a binary tree on $[m_2]$. More generally, there is an edge from each vertex of the binary trees on $[m_i]$ to the root of a binary tree on $[m_{i+1}]$. A binary search tree on $[m]$ can thus be viewed as a tree whose nodes are each associated with a dyadic range in $[m]$. The source s of G_{RT} is such that $s.range = \mathcal{D}$. For a node v of a binary tree on $[m_i]$, let $v.dyadic$ denote the dyadic range in $[m_i]$ that v is associated with. Let $w.range = w_1 \times \cdots \times w_d$ be the canonical range of the root w of v's binary subtree on $[m_i]$. We have

$$v.range = w_1 \times \cdots \times w_{i-1} \times v.dyadic \times [m_{i+1}] \times \cdots \times [m_d] \quad (1)$$

See Figure 2a for an example of 2-dimensional range tree.

Algorithm 3: $\mathsf{BRC}_{RT}(T, q, v)$

```
1:  // Invoked with BRC_RT(T, q, s), where s is the source of T
2:  W ← {v}
3:  q_1 × ··· × q_d ← q
4:  for i ∈ [d] do
5:    W' ← ∅
6:    for w ∈ W do
7:      w_1 × ··· × w_d ← w.range
8:      q̂ ← w_1 × ··· × w_{i−1} × q_i × [m_{i+1}] × ··· × [m_d]
9:      Let T̂ ⊆ T be the subtree on [m_i] rooted at w.
10:     W' ← W' ∪ BRC(T̂, q̂, w)
11:   W ← W'
12: return W
```

The multi-dimensional range tree for $d > 1$ does not satisfy the properties of the tree in Theorem 2. The multi-dimensional range tree can be viewed as being composed of subtrees that subdivide the domain along different dimensions; these subtrees satisfy the properties of Theorem 2. We thus design a ***best range cover*** for multi-dimensional range trees, Algorithm 3 (BRC_{RT}), that calls Algorithm 1 (BRC) as a subroutine on these subtrees.

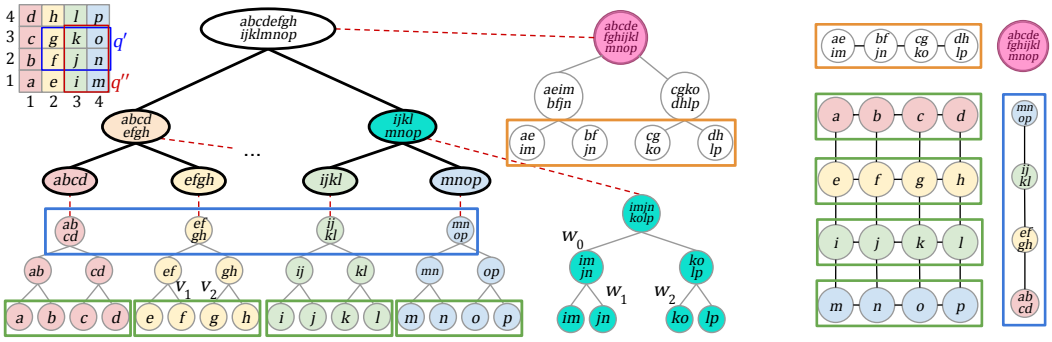
Theorem 5. *Let G_{RT} be a multi-dimensional range tree. For any range query q, the cover returned by BRC_{RT} has no false positives and is the unique minimal cover.*

Observe that for queries of the same size, BRC_{RT} can produce covers of different sizes. As such, cover size may reveal some information about the location of the queried range. Kiayias et al. [35] introduce the notion of a ***uniform range cover*** that seeks to resolve this problem by making the size of the tokenset depend only on the size of the range, and not on its location in the domain. Let URC denote the uniform range cover algorithm for 1D ranges from [35]. It takes as input a 1D range tree G_{RT}, a range query q, and the source node of s, and returns the uniform range cover of q in G_{RT}. We extend the 1D URC algorithm to higher dimensions. Our uniform range cover algorithm for multi-dimensional range trees, denoted URC_{RT}, is identical to Algorithm 3 (BRC_{RT}) except that on line 10, it calls URC as a subroutine instead of BRC.

Theorem 6. *Let G_{RT} be a range tree on $[m_1] \times \cdots \times [m_d]$ and σ be any permutation on $[d]$. If q is a range query of size $R = R_1 \times \cdots \times R_d$ and q' is a range query of size $R = R_{\sigma(1)} \times \cdots \times R_{\sigma(d)}$, then their respective URC_{RT} covers W and W' are such that $|W| = |W'|$.*

4.3 QDAG SRC Scheme

We present the ***QDAG SRC scheme***, which is derived from the quadtree, supports single range covers at the expense of $O(R^d)$ false positives, and extends the 1D TDAG scheme [13] to higher



(a) Two-dimensional range tree on a 4×4 domain. (b) Attack on the range tree URC scheme.

Figure 2: Range tree scheme and attack on the URC version. (a) The binary tree over the first dimension is shown with thick black lines and the binary trees over the second dimension are shown with thin gray lines. Using BRC, query range $q' = [2,4] \times [2,3]$ (in blue) corresponds to cover $\{v_1, v_2, w_1, w_2\}$ and query range $q'' = [3,4] \times [1,3]$ (in red) corresponds to cover $\{w_0, w_2\}$. The nodes in the green/orange/blue rectangles and in pink correspond to tokensets of size 1, under URC. (b) Graph G built by Algorithm 5. Its nodes are tokensets of size 1 (e.g. (a), $(mnop)$), and its edges correspond to tokensets of size 2 (e.g. (a, b), $(\{mnop\}, \{ijkl\})$). The largest component of G is the grid of the domain points.

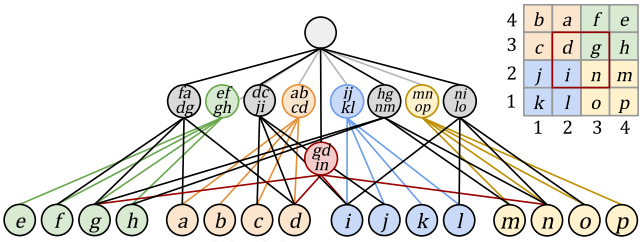


Figure 3: QDAG for a $[4] \times [4]$ domain obtained by augmenting the quadtree by adding the dark gray and red nodes.

dimensions. The response size overhead is an inherent limitation of schemes that index the domain using only hypercube ranges.

Given a multi-dimensional database D with domain $\mathcal{D}$ we build the data structure the bottom up, starting with m leaves corresponding to points of domain $\mathcal{D}$. At level j we add nodes for all hypercubes of size 2^{n-j} tiling the domain, as well as each of these hypercubes shifted by 2^{n-j-1} along each dimension. For each node at level j, we add directed edges to all nodes in level $j-1$ which it covers. We recursively build the structure until we reach the (source) root node that corresponds to the entire domain. Each node v in this DAG is associated with a d-dimensional hypercube. To execute a range query, the client computes its SRC cover.

The QDAG SRC scheme is illustrated in Figure 3. The $O(R^d)$ false positive rate, $O(m)$ space usage, and overall complexity of the QDAG SRC scheme are described in detail in Appendix D.3.

4.4 Complexity and Leakage

Table 1 compares the schemes in this section. The SRC scheme has optimal query size but allows false positives. The other schemes avoid false positives but incur query size overhead. To achieve efficient client query time, the range cover algorithm builds the tokenset without instantiating the scheme's DAG, which is implicitly defined by the parameters of the domain. Thus, we can assign IDs to the nodes of the DAG so that the ID of each node in the cover is computed in $O(1)$ amortized time and space. Hence, the query time and space at the client is proportional to the query size (to generate the tokenset) plus the response size (to decrypt the received response). The query execution time at the server is also proportional to the query size plus the response size, since accessing the partial response associated with a token takes $O(1)$ expected time with an efficient multimap implementation. The client space is $O(1)$, plus $O(\log^d m)$ temporary space for the range tree schemes when a query is issued, and temporary space proportional to the response size when the response is received.

Theorem 7. *Let Σ be an EMM scheme that leaks search and volume pattern. If the linear LRC, range tree BRC/URC, and QDAG SRC schemes are instantiated with Σ, then each of these range encrypted multimap schemes leaks search, volume and structure pattern.*

Note that the structure pattern of the QDAG SRC scheme gives no additional information beyond its search and volume pattern.

5 LEAKAGE ANALYSIS AND ATTACKS

We explore the information theoretic limitations on what can be recovered from the leakage profiles of our schemes and present reconstruction attacks. Our work shows that volume and search pattern leakage combined with structural leakage can be as exploitable as access and search pattern leakage.

Preliminaries. As before, we consider a d-dimensional database D of size n over domain $\mathcal{D}$ of size m. We assume that $\mathcal{D}$ is encrypted with one of the schemes presented in Section 4. The tokenset, $\mathbf{t}$, of a query q is the set of tokens associated with q. For each token t sent by the client, the server returns the encrypted set $C(t)$ retrieved from an encrypted multimap (Definition 5), from which the adversary determines the volume, vol_t, associated with token t. For each scheme, we present a reconstruction attack that takes as input a ***volume map***, denoted with VM, that for each tokenset $\mathbf{t}$, maps $\mathsf{VM}[\mathbf{t}] = \sum_{t \in \mathbf{t}} vol_t$, and for each token $t \in \mathbf{t}$, maps $\mathsf{VM}[t] = vol_t$.

Our attack on the SRC scheme also takes as input a ***frequency map*** FM, which associates each tokenset with the number of times the it has been observed. Maps VM and FM take linear time to build on the size of the input and require less storage than the input, since their sizes are independent of n. We assume the adversary has knowledge of m, d, n, as well as of the range encrypted multimap scheme employed. Our attacks take as input VM and (in the SRC case) FM and return a grid comprising one node for each point

of in domain $\mathcal{D}$, where each node is labeled with the number of database records at the corresponding point.

We show that VM and FM are an equivalent representation of the multiset of structure pattern. We assume that the queries are issued independently; we do not exploit the order of the queries, for example, by assuming that their order is correlated to their position. It is thus sufficient to consider a multiset of the structure pattern.

Theorem 8. *Let Σ be an EMM scheme leaking search and volume pattern. Let D be a d-dimensional database over domain $\mathcal{D}$ and let $q^{(1)}, \ldots, q^{(\ell)}$ be range queries over $\mathcal{D}$. Then there exists an invertible transformation between the multiset of leakage $\{\{\mathsf{Str}(D, q^{(i)})\}\}_{i\in[\ell]}$ and the corresponding volume map* VM *and frequency map* FM.

Equivalent Databases. We first generalize the notion of *equivalent databases* from [17, 40] below. Intuitively, two databases are $\mathcal{L}$-equivalent if they are indistinguishable from their leakage alone.

Definition 6. *Let D and D' be databases with domain $\mathcal{D}$ and the same record IDs. Let $\mathcal{L} = (\mathcal{L}_S, \mathcal{L}_Q)$ be a leakage function and $\mathbf{Q}$ be the set of range queries on $\mathcal{D}$. Databases D and D' are **$\mathcal{L}$-equivalent** if $\{\mathcal{L}(D, q)\}_{q\in\mathbf{Q}} = \{\mathcal{L}(D', q)\}_{q\in\mathbf{Q}}$. The set of equivalent databases is called the* reconstruction space.

5.1 Leakage Analysis of the Linear Scheme

In the linear scheme (Section 4.1), each tokenset comprises a token for each point in the queried range. Each domain point is associated with a unique token across all queries. Thus, the linear scheme leaks the size of the query range and information about the points in the range. When the adversary observes a tokenset of prime size, p, they can infer that the range has size p in one dimension and size 1 in the other dimensions. This leakage allows the adversary to extract useful single-dimensional information.

Reconstruction Space.

Theorem 9. *Let D be a database on a d-dimensional domain and let $\mathcal{L}$ be the leakage of the linear scheme. The set of databases $\mathcal{L}$-equivalent to D, or reconstruction space of D, corresponds to the symmetries of a d-cube. (i.e. rotation/reflection across each axis).*

Reconstruction Attack. Our reconstruction attack finds queries of prime size, groups the search tokens into one-dimensional segments and then orders them. Our attack follows in five steps:

(1) **Queries of prime size.** We find all queries of prime size.
(2) **Group one-dimensional sections.** Note that if the intersection of two tokensets of prime size has at least two search tokens, then these queries must be from the same one-dimensional section (e.g., same row or column). We create a map 1dSlices, where a key-value pair corresponds to a set of search tokens mapping to a set of tokensets, where all the search tokens in a key are on the same one-dimensional section.
(3) **Order one-dimensional sections.** We use PQ-trees [4] to get the partial order of the search tokens in each key of 1dSlices.
(4) **Order Reconstruction.** We construct a graph G whose nodes are the observed tokens. For each PQ-tree, we find a frontier (a possible order of the tokens), and use it to add edges in G between neighboring tokens in each frontier.

Algorithm 4: *LinearReconstruction*(VM)

```
// Find tokensets that correspond to one-dimensional queries.
Let primeTokensets store the tokensets of unit and prime size in VM.
Let 1dSlices be an empty map, mapping tokens (which share the same
  coordinate in one dimension) to a list of tokensets
// Group tokensets by one-dimensional section.
for each tokenset t in primeTokensets do
    Find all keys, K, in 1dSlices that intersect in ≥ 2 elements with t
    Add t to K and let V be a list of the values of K in 1dSlices
    Delete all keys in K from 1dSlices and add K → V to 1dSlices
// Order the elements of each one-dimensional section.
Create a PQ-tree for each key of 1dSlices with its values.
// Make a grid representing the domain value of each search token.
Let G be a graph with nodes all the observed search tokens.
for each PQ-Tree T do
    Pick a frontier (a possible ordering of the search tokens) of T.
    Add an edge to G for every pair of neighbors in this frontier.
// Reconstruct the database.
Label the nodes of G with their volume in VM.
return G
```

(5) **Database Reconstruction.** Since we know the volumes that correspond to each search token and the domain point each search token corresponds to (as we have ordered them), we achieve full database reconstruction.

Algorithm 4 shows our reconstruction attack on the linear scheme.

Theorem 10. *Let D be a database over a d-dimensional domain $\mathcal{D} = [m_1] \times \cdots \times [m_d]$ of size m and let D be encrypted with the linear scheme. Given the volume map for a set of range queries on D comprising all queries of unit and prime size, Algorithm 4 achieves full database reconstruction of D by building in $O(m^5)$ time and $O(m^3)$ space an $O(m)$-size representation of the reconstruction space of D. The input to the algorithm is available with probability $1 - \frac{1}{m^2}$ after*

$$O\left(\sum_{i=1}^{d} \frac{m^2}{m_i} \log m_i \cdot \log\left(\frac{m^2}{m_i} \log m_i\right)\right) \tag{2}$$

uniformly distributed queries, which is $O(m^{2-\frac{1}{d}} \log^2 m)$ queries when $m_i = m^{1/d}$ for $i = 1, \cdots, d$.

For the case of 2D range queries, we can transform the leakage of the linear scheme into access and search pattern leakage and give it as input to the FDR attack on 2D databases from [17]. This transformation results in information loss, making the reconstruction space potentially exponential in the number of records.

5.2 Leakage Analysis of the Range Tree Scheme

For the range tree scheme, the client issues queries that are expanded into sub-queries associated with nodes of the range tree. In this section, we present and analyze concrete attacks on the range tree scheme under the URC and BRC range covering techniques.

In their journal paper, Demertzis et al. [14] mention that, for the 1D range tree BRC and URC schemes, the structural leakage of queries from overlapping ranges allows the adversary to exploit the co-occurrence of tokens and reconstruct the tree structure (and hence the database). For this reason, they suggest using such schemes for the restrictive scenario where no two queries issued

Algorithm 5: *RangeTreeReconstructionURC*(VM)

1: Let Q_1 be the keys of VM of size 1.
2: Let Q_2 be the keys of VM of size 2 with only members of Q_1.
3: Construct graph G with nodes the elements of Q_1
4: **for t** = $(t_0, t_1) \in Q_2$ **do**
5: Add an edge between t_0 and st_1 in G.
6: Label the nodes of G with their volume in VM.
7: **return** the largest connected component of G

by the client overlap. Notably, they do not give the details of such an attack, address its asymptotic performance, or implement it.

Reconstruction Space under URC.

Theorem 11. *Let D be a database with domain $\mathcal{D} = [m_1] \ldots \times [m_d]$ and let $\mathcal{L}$ be the leakage of the range tree scheme with range covering algorithm* URC. *The set of databases $\mathcal{L}$-equivalent to D corresponds to the symmetries of a d-cube.*

Reconstruction Attack under URC. For this attack, we leverage the fact that URC leaks neighboring point-value search tokens. Thus, if we can identify such queries, we can determine not only which tokens correspond to point queries, but also their neighbors. We first make the following observations:

- Under URC, if a client queries a query q with tokenset of size 1, then they are querying either everything or one point. In two dimensions, there are four cases:
  (i) Query q is a point query (green rectangles in Figure 2b).
  (ii) Query q is a row query (orange rectangles in Figure 2b).
  (iii) Query q is a column query (blue circles in Figure 2b).
  (iv) Query q queries the whole database (pink circle in Figure 2b).
- Let q be a query with tokenset (t_0, t_1). The union of t_0's and t_1's corresponding range must form a hyper-rectangle.
- Under URC any range of size 2 corresponds to two tokens.

We sketch Algorithm 5: Let Q_1 be the set of queries with tokenset of size 1. In Figure 2a, these are the nodes in a box or highlighted. Create a graph G with nodes Q_1. Add an edge (t_0, t_1) in G, if there exists a tokenset (t_0, t_1). The largest connected component of G corresponds to the ordered search tokens of the database. Figure 2b contains the corresponding graph G. We complete the attack by mapping each search token of G to its corresponding volume.

Theorem 12. *Let D be a database over a d-dimensional domain of size m and let D be encrypted with the range tree scheme and uniform range cover* (URC). *Given the volume map for all range queries on D, Algorithm 5 achieves full database reconstruction of D by building in $O(m^2 \log^d m)$ time and space an $O(m)$-size representation of the reconstruction space of D. The input to the algorithm is available with probability $1 - \frac{1}{m^2}$ after $O(m^2 \log m)$ uniformly distributed queries.*

Reconstruction Space under BRC.

Theorem 13. *Let D be a database with domain $\mathcal{D} = [m_1] \times \cdots \times [m_d]$ and let $\mathcal{L}$ be the leakage of the range tree scheme with range covering algorithm* BRC. *The set of databases $\mathcal{L}$-equivalent to D corresponds to the symmetries of a d-cube.*

Algorithm 6 returns a representation of the reconstruction space of D, which contains only the symmetries of the square.

Reconstruction Attack under BRC. We present an attack on the range tree BRC scheme that achieves polynomial run-time (a sketch of our attack can be found in Figures 4 and 5). We first define the following terms. A *leaf node* is a node that has no children. A *boundary node* corresponds to a query that covers at least one extreme domain value. For example, nodes a and p are boundary nodes in Figure 4(a). A node that is not a leaf node or a boundary node is an *inner node* (orange in Figure 4(a)).

First, we find all distinct queries that are mapped to a tokenset of size 2 and generate a co-occurrence graph $G = (V, E)$ where nodes V correspond to tree nodes and edges E denote that the endpoints form a tokenset (Figure 4(b)).

Infer the inner nodes. We first identify the inner nodes in the range tree (orange in Figure 4(a)). Given query tokensets (s_1, s_2, s_3), (s_1, s_2) and (s_2, s_3), there can be no query that maps to (s_1, s_3). Thus, s_2 can be identified as an inner node (see Figure 4 for an example).

Trim the co-occurrence graph. At this point, we want to distinguish between the boundary and leaf nodes. We observe that leaf nodes form triangular structures in G with their parent nodes (e.g. *c-ab*, *b-c* and *b-cd* in Figure 4(a)). We remove any edges between inner nodes in G. Additionally, we use the number of times two tokens appear in a tokenset together i.e. the edgecounts, to remove any edges with edgecount more than two. This is because parents of leaf nodes have an edgecount of two with one of their children, but ancestors further up the tree have a higher edgecount.

Inner Grid Reconstruction. We now use the co-occurrence graph G along with the inner nodes and triangular structures to identify the leaf nodes in each 1D range tree. To distinguish between leaf and non-leaf nodes that look identical in G, we use the original co-occurrence graph. We now observe a component in G' that contains a d-dimensional grid (Figure 4(d)) containing nodes that form the triangular structures. Since we know the relationship of the nodes in triangular structures, we remove the inner nodes (Figure 4(c)) and re-order the records.

Inferring the extreme nodes' volumes. So far we have determined the volumes of each non-extreme domain point. We now extrapolate the volumes of domain points extreme in only one dimension. In our grid structure above (Figure 4(d)), we can see that there are some nodes that are inner nodes. Replacing the inner nodes with a volume (the inner node's volume minus the volume of its neighbor), we can reconstruct the volume of these domain points. We add an edge between two such nodes, if the inner nodes they replace were neighbors in the original co-occurrence graph G_o.

Then, for each dimension $i \in [2, d]$ in increasing order, we identify all missing volumes on the grid that are on extreme domain values in i dimensions. For each such volume v (see. e.g., example the corner represented by a in Figure 5(c)), we identify $i + 1$ tokens that surround the i-cube of size 2^i whose corner is v. Then, we find the smallest tokenset that contains these $i + 1$ tokens. It contains one more token corresponding to the i-cube. Since we know the volumes of all the points but a, we can extrapolate a's volume.

Once we identify all volumes of nodes on extreme domain values in i dimensions, we add the relevant grid edges, based on the new nodes' locations on the grid (if necessary).

Theorem 14. *Let D be a database over a d-dimensional domain of size m and let D be encrypted with the range tree scheme and best*

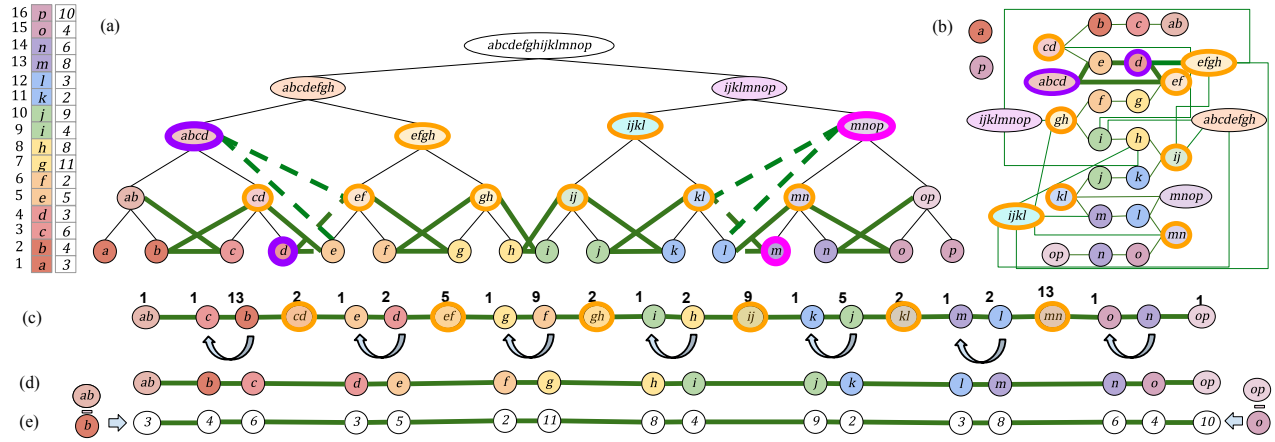


Figure 4: Attack on the range tree BRC scheme (Algorithm 6) for a 1D domain. (a) Domain with volume of each point and range tree. We find the inner nodes of the range tree (orange) by relying on the property that tokensets form a continuous range (Line 12). E.g, tokensets $(d, \{ef\})$, $(\{ef\}, g)$ and $(d, \{ef\}, g)$, and the absence of tokenset (d, g) imply that $\{ef\}$ is an inner node. Thick green lines show the triangular structures identifying leaf nodes. (b) Co-occurrence graph G, whose edges (in green) join nodes of the range tree that form a tokenset, e.g., (b, c), (Line 2). (c) Construction of graph G_{trim}. We remove from G edges between inner nodes (Line 13). We use the times two tokens appear in a tokenset together (edgecounts) (Line 16) to remove edges with edgecount > 2 (Line 20). We identify most leaf nodes using G, but some nodes like $\{abcd\}$ and d appear identical in G (dashed green edges in part a). We distinguish them using graph G, e.g., $\{abcd\}$ has fewer edges than d (Line 22). Graph G_{trim} now contains all inner nodes from G with edgecount 2, and some non-inner neighbors. (d) We extract the inner nodes from the new graph (Line 25), and swap every other pair of nodes (Line 30). (e) We replace the two nodes with only one edge ($\{ab\}$,$\{op\}$) with their volume minus the volume of their neighbor (Algorithm 7, line 2). We reconstruct the number of database records at all domain points by assigning volumes to the remaining nodes of the graph (Algorithm 7, line 15).

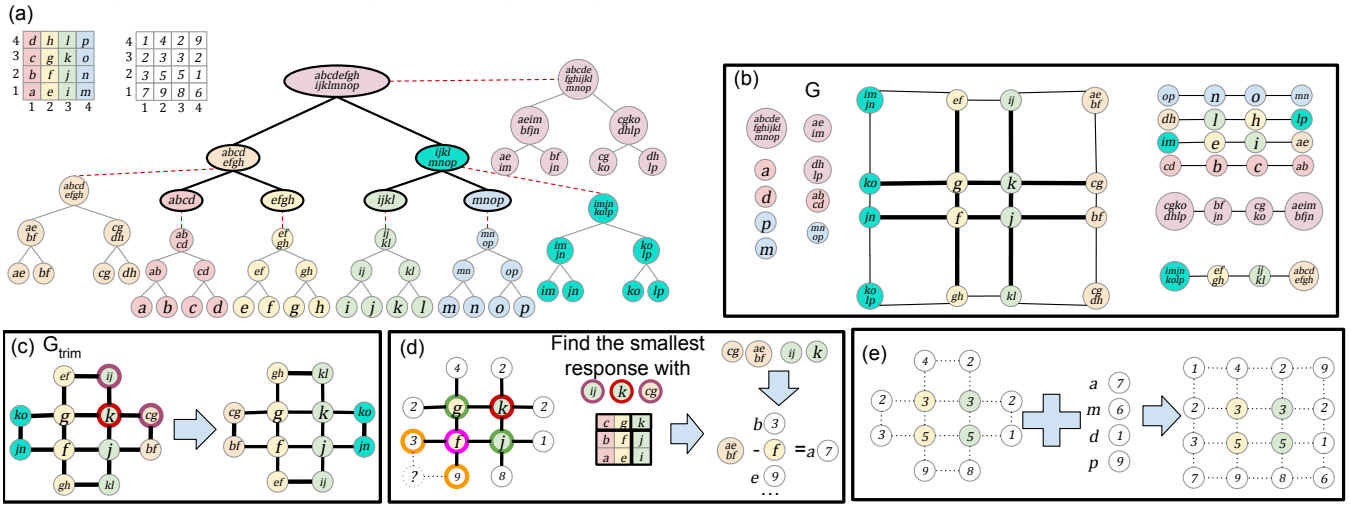


Figure 5: Attack on the range tree BRC scheme for a 2D domain. (a) Domain with volume of each point and range tree. (b) The attack works similarly to the 1D case, creating the co-occurrence graph G. (c) Then, we similarly create G_{trim}. (d) Algorithm 7 extrapolates the volume at every domain point. We can find any volumes on a domain point that is extreme only in 1D by replacing each such non-leaf node in G' with its volume minus the volume of its neighbor (Line 2). For example, we find the volume at e by subtracting the volume of f from ef. For each missing volume (domain values extreme in 2 dimensions), say the volume at a, we find node k, diagonal to a and 2 away in each dimension that a is extreme in. We then identify two neighbors of k in G_{trim}, $\{ij\}$, $\{cg\}$, such that the smallest tokenset (Line 12) containing k, $\{ij\}$ and $\{cg\}$ contains $\{aebf\}$, the token corresponding to a 2x2 square that contains a. Since we know all the volumes but for a's, we can extrapolate the volume of a (Line 13). (e) We similarly identify the volumes of all the corner nodes (a, d, p, m), combine them with the augmented grid and reconstruct the database (Line 15).

range cover (BRC). *Given the volume map for all range queries on D, Algorithm 6 achieves full database reconstruction of D by building in $O(m^4)$ time and $O(m^2 \log^d m)$ space an $O(m)$-size representation of the reconstruction space of D. The input to the algorithm is available with probability $1 - \frac{1}{m^2}$ after $O(m^2 \log m)$ uniformly distributed queries.*

5.3 Leakage Analysis of SRC Schemes

The SRC variant is considerably more difficult to attack than URC and BRC variants as SRC queries contain only a single (encrypted) range cover. This prevents us from making the same spatial connections between queries that enabled the prior attacks. In fact, Demetrzis et al. [14] conjecture that even novel attacks could not

Algorithm 6: *RangeTreeReconstructionBRC*(VM)

Let E, Q be the keys (tokensets) of VM of size 2 and ≥ 2, respectively.
Construct undirected graph G that connects tokens that appear as pairs in E.
Let $G_o = G$
// Remove any "inner" edges from G
Initialize set inner $\leftarrow \emptyset$.
Initialize table edgecounts with edgecounts$[e] = 0, \forall e \in E$.
for each tokenset $S \in Q$ **do**
  Construct subgraph G_S of G induced by the nodes of S.
  // Graph G_S is a hyperrectangle (Lemma 4)
  Let C be subset of nodes of G_S with the smallest degree in G_S.
  Let $I \leftarrow S - C$ // I is a subset of inner nodes of S
  Add I to set inner.
  Remove any edges $\in G_S$ from G not connected to a node in C.
  if $|C| = 2$ // We may be in a one dimensional slice. **then**
    for each edge $e \in G_S$ **do**
      edgecounts$[e] \leftarrow$ edgecounts$[e] + 1$
// Disambiguate identical components of the graph
for all node $v \in$ inner **do**
  if there is no edge e incident on v where edgecounts$[e] = 2$ **then**
    Remove node v from G and from inner.
  else
    Find all neighbors of v in G with edgecounts$[(v, u)] = 2$ and remove them from G, but for one with the most edges in G_o.
Let G_{trim} be the largest component of G
// Contract edges between remaining inner nodes
for each vertex $u \in$ inner **do**
  Let v, w be the neighbors of u in G.
  Add edge (v, w) to G, and remove node u from G.
Let G' be the subset of G containing all nodes with more than one neighbors. // G' contains multiple grids of various dimension.
for each dimension i **do**
  On each one-dimensional section of G' (e.g. row, column...) spanning dimension i, swap every other pair of nodes, skipping the first one.
Add back any nodes not in G' from G.
return *FindExtremeVolumes*(VM, G_o, G_{trim}, G') (Algorithm 7)

Algorithm 7: *FindExtremeVolumes*(VM, G_o, G_{trim}, G')

// Find volumes of extreme domain points
Replace any node with one neighbor in G' with its volume minus its neighbors' volume.
Add an edge between two new volume nodes, if the nodes they replaced were connected in G_o.
Let G' consist only of its largest component, a d-dimensional grid missing some nodes.
for $i \in [2, d]$ **do**
  for nodes v in G' missing a volume, extreme in any i dimensions **do**
    Let N_v be the potential neighbors of v in G'.
    Let c be the common neighbor of N_v in G' (not v).
    Create N'_v by finding the other (leaf) neighbors of c in G' in the same dimension as each node in N_v.
    Find the other common neighbor of N'_v in G' that is not c, c'
    Create N''_v by finding the other (non-leaf) neighbors of c' in G_{trim} in the same dimension as each node in N'_v.
    Find the smallest key, k, in VM that contains c' and N''_v.
    Let v's volume be the sum of the volumes of all nodes in k minus the volumes of N_v, N''_v, c and c'.
  Add relevant edges for the new volume nodes based on their location on the grid in G'.
Label the nodes of G' with their volume in VM.
return G'.

Algorithm 8: *GenericReconstructionSRC*(VM, FM)

Let max be the maximum volume in FM.
Let F be the set of frequencies in FM.
Let G be the underlying DAG and for each node $v \in G$, create integer ILP variable x_v with bounds $[0, \text{max}]$.
for non-leaf node $v \in G$ **do** add Equation 3 to the ILP.
for $f \in F$ **do** add Equations 4 to the ILP.
Run the ILP solver to retrieve assignment A.
Let H be a grid corresponding to the tokens forming leaves of the tree.
Label the nodes of H with their volume in VM.
return H

achieve full database reconstruction against their 1D SRC schemes. Nevertheless, we show that we can reliably attack SRC schemes.

Reconstruction Space under SRC. The QDAG SRC scheme is the only scheme from Section 4 with a reconstruction space that displays symmetries other than those of the d-cube. We present a database that demonstrates these additional symmetries (Appendix E.7), which yields the following lower bound.

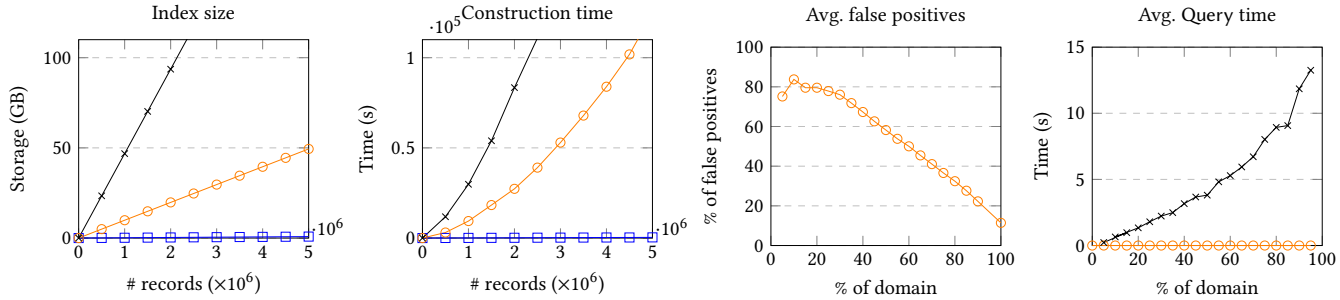
Theorem 15. *Let D be a dense database with domain $\mathcal{D} = [m_1] \times \cdots \times [m_d]$ and let $\mathcal{L}$ be the leakage of the QDAG scheme with range covering algorithm* SRC. *Let $S_{\mathcal{L}}$ be the set of databases $\mathcal{L}$-equivalent to D. We have $|S_{\mathcal{L}}| \geq 2^{d+2^{(d-1)}}(d)(d!)$.*

Reconstruction Attack under SRC. Our SRC attack generalizes to a broad class of SRC schemes. Let (G, SRC) be a range-supporting data structure satisfying the following two properties: (1) Every non-sink node v in G has a subset of children C, such that $\{c.range : c \in C\}$ partition $v.range$, and (2) sinks of G are 1-1 with the domain point values. Our SRC attack works on all range encrypted multimap schemes built with such a (G, RC) pair.

To attack these schemes, we construct and solve an integer linear program (ILP) whose constraints are based on the underlying DAG; The ILP is satisfied by any database in the reconstruction space, given that every possible range query has been issued exactly once. For every node v in the DAG (e.g. the QDAG) we associate a variable x_v that corresponds to the volume of $v.range$. For each non-sink v in G we write the following constraint:

$$\boxed{x_v = \sum_{c \in C} x_c} \tag{3}$$

where C is the set of v's children whose canonical ranges partition $v.range$. Now suppose that every domain query has been issued exactly once. We can determine exactly how many unique queries correspond to each SRC node in G. We refer to this as the *frequency* of the node. Let F be the set of all frequencies. For a frequency $f \in F$, let X_f be the set of variables corresponding to nodes with frequency f and let V_f be the set of volumes with frequency f. For each $f \in F$, we restrict the variables in X_f to values in V_f, since there should be a 1-1 correspondence between variables in X_f and volumes in V_f. We implement the correspondence as follows. For each $f \in F$ we define a $|X_f| \times |X_f|$ matrix of Boolean variables $b_{1,1}, b_{1,2}, \ldots, b_{|X_f|,|X_f|}$ such that each row corresponds to a variable in X_f and each column corresponds to a volume in V_f. For each



(a) **(b)** **(c)** **(d)**

Figure 6: (a) Index size and (b) construction time of the linear (—□—), range tree BRC/URC (—×—) and QDAG SRC (—○—) schemes for the Gowalla dataset. (c) Average false positive rate for the QDAG SRC scheme as a function of the size of the queried range, as computed by sampling 100,000 random queries. (d) Average query time at the server on the Spitz dataset as a function of the size of the queried range, as computed by sampling 100,000 random queries. The linear scheme is not included due to its prohibitive query size (i.e., tokenset size).

frequency $f \in F$, we then write the constraints

$$x_s - \sum_{t=1}^{|X_f|} v_t b_{s,t} = 0; \quad \sum_{s=1}^{|X_f|} b_{s,t} = 1; \sum_{t=1}^{|X_f|} b_{s,t} = 1 \tag{4}$$

where $x_s \in X_f$ and $v_t \in V_f$.

Our attack either needs to observe every range query exactly once or needs knowledge of the query distribution. Given the distribution, after observing enough queries, the adversary can deduce how many unique queries correspond to each tokenset. The adversary can then create constraints using Equations 3 and 4 and use a generic ILP solver to reconstruct the database. Algorithm 8 takes as input VM and FM and returns grid graph G whose nodes are labeled with volumes.

Theorem 16. *Let* (G, SRC) *be a range-supporting data structure for a* d*-dimensional domain* $\mathcal{D}$ *such that:*

(1) each non-sink v *in* G *has a subset of children* C *such that their canonical ranges,* $\{c.range : c \in C\}$*, are a partition of* $v.range$*;*
(2) the sinks of G *are in 1-1 correspondence with the points in* $\mathcal{D}$*.*

Let D *be a database over* $\mathcal{D}$ *encrypted with* GenericRS *using* (G, SRC) *and an EMM scheme which leaks volume and search pattern. Given the volume map and frequency map for all range queries on* D*, where each query is issued exactly once, Algorithm 8 achieves full database reconstruction of* D*. The input to the algorithm is available with probability* $1 - \frac{1}{m^2}$ *after* $O(m^4 \log m)$ *uniformly distributed queries.*

Note that the effectiveness of Algorithm 8 depends on the size of the reconstruction space, which is determined by the underlying range-supporting data structure (G, SRC) and the database D.

Our attack on SRC schemes is related to the attack by Kornaropoulos, Papamanthou, and Tamassia (KPT) [37], which approximately reconstructs a database from *one-dimensional* range queries. The KPT attack utilizes *counting functions* to determine the number of canonical ranges that return a given (encrypted) response. This information is used to build a system of equations that captures the distance between consecutive records. In contrast, we build a system of equations representing how the volume of canonical ranges is distributed to its subranges, as given by the DAG. Our attack assumes a uniform query distribution to observe all possible queries with the same frequency and aims at full database reconstruction. The KPT attack does not assume knowledge of the query distribution and uses nonparametric estimators over a subset of the possible queries to achieve an approximate reconstruction.

6 EXPERIMENTS

In this section, we experimentally evaluate the performance of our Linear, Range-BRC/URC and QDAG-SRC schemes and attacks on them using the following real-world datasets:

Cali [39]: 21,047 latitude-longitude points of road network intersections in California, a dataset used in a prior 2D attack [40].

Spitz [52]: 28,837 latitude-longitude points of phone location data of politician Malte Spitz between August 2009 and February 2010 a dataset previously used in several reconstruction attacks [17, 36, 40].

Gowalla [11]: 6,442,892 latitude-longitude points from users of the Gowalla social networking website between 2009 and 2010, a dataset used in the experiments by Demertzis et al. [13]. We further replicate Demertzis et al.'s Gowalla experiments by randomly partitioning the dataset into 10 sets, each consisting of 500,000 records. We then measured the indexing time and cost of our schemes by increasing the size of the domain by a new set of 500,000 tuples.

Schemes. For each dataset, we used the latitude and longitude as query attributes and 16-byte random strings as records. For our scheme experiments, we normalized the domain of Cali and Spitz to $[2^{10}] \times [2^{10}]$ and the domain of Gowalla to $[2^{16}] \times [2^{16}]$ to have the same number of domain points as in the 32-bit domain used by Demertzis et al. [13].

Figure 6 shows the performance of our schemes. Figure 6(a) gives the index size (space), which is proportional to the number of database records. Schemes with more canonical ranges have longer build times (see Figure 8). For example, the Range-BRC/URC scheme has more range covers than the QDAG-SRC scheme, and thus has larger index size. Figure 6(b) shows how the construction time increases with the number of records and the index size. Figure 6(c) gives the average number of false positives for the QDAG-SRC scheme as a function of the query range size (expressed as percentage of the domain size). In Figure 6(d), the query time grows with the range size in the Range-BRC/URC schemes, but remains flat in the QDAG-SRC scheme.

Attacks. We performed experiments on our attacks on the Linear, Range-URC, Range-BRC and QDAG-SRC schemes. Our attacks

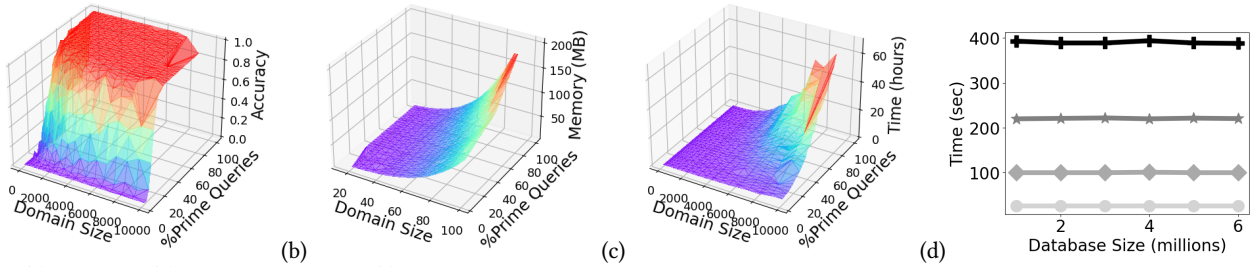


Figure 7: (a) Accuracy, (b) memory usage, and (c) runtime of our attack on the linear scheme for 2D databases of different domain sizes, after observing different percentages of prime-size queries. (d) Runtime of the attack for the Gowalla dataset, varying the number of records (in millions) and percentage of prime-size queries (25% (lightgray circle •), 50% (gray diamond ♦) , 75% (darkgray star ⋆), plus sign 100% (black +)).

always returned the original database up to the symmetries of a d-cube, when given the complete input.

Figure 7 shows our experiments on the Linear attack. In Figure 7(a) we show the median accuracy of our attack under 2D databases of different domain sizes, after observing different percentages of the prime queries. We measure the accuracy of our attack as the percent of correctly reconstructed domain point volumes. We observe that this attack achieves great accuracy with a relatively small percent of the prime queries. The attack requires little storage space (Figure 7(b)), but as the domain size increases it can take more time (Figure 7(c)). We also ran our attack against a $[2^5] \times [2^5]$ Gowalla dataset, varying the number of records from 1 million to 6 million. This shows that our attack's runtime is not affected by the number of records; only the domain size.

Recall that for the case of 2D range queries, we can transform the leakage of the linear scheme into access and search pattern leakage. This leakage can be given as input to the approximate database reconstruction attack on 2D databases from [40]. However, this transformation results in loss of information, which makes the reconstruction space potentially much larger (exponential in the number of database records).

Scheme	Cali		Spitz	
	Index Size (MB)	Build Time (s)	Index Size (MB)	Build Time (s)
Linear	3.41	6.23	4.67	2.07
Range-BRC/URC	985.38	1095.20	1350.09	562.86
QDAG-SRC	207.99	208.30	284.96	118.70

Figure 8: Scheme costs for the Cali and Spitz datasets.

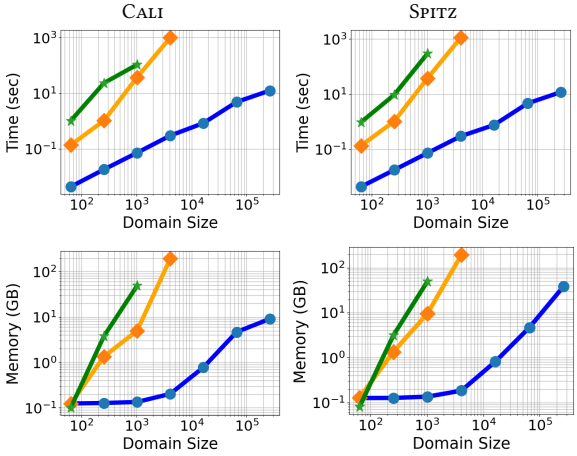


Figure 9: Median runtime in seconds (top) and median memory usage (GB) (bottom) of our attacks on the range tree URC (blue circle •), range tree BRC (orange diamond ♦), and QDAG SRC (green star ⋆) schemes for the Cali and Spitz databases on different domain sizes.

We also ran our attacks on the Range-URC, Range-BRC and QDAG-SRC schemes. We used the Spitz and Cali dataset normalized at different domain sizes to showcase our attacks in Figure 9. We observe that the QDAG-SRC attack takes the longest time and generally requires more memory. This attack needs to solve an integer linear programming problem, and thus has a longer runtime. The Range-URC attack is the most efficient. We also run these attacks against the Gowalla dataset to observe how the number of records affects the runtime and memory requirements (Figure 10). We observe that the attacks are generally not affected. The QDAG-SRC attack against the Gowalla dataset (Figure 10) shows some random variance in the runtime and memory needed. We believe this is due to randomness in the solver that causes it to take different search paths on each execution.

Implementation Details. We implemented our schemes and the URC, BRC, and SRC attacks in Python 3.9.2 and the Linear attack in C++. The Linear attack utilizes a C++ library that implements PQ-trees [22]. We ran all of our experiments on a compute cluster. For simplicity, we used the same compute node for the client and the server so our results do not include any latency that would be incurred due to network transmission.

For cryptographic primitives, we used version 3.4.7 of the Python cryptography library [48]. To match the evaluation of Demertzis et al. [13], we use SHA-512 for PRFs and AES-CBC (with 128-bit block size) for encryption. For our underlying EMM scheme, we used our own implementation of the Π_{bas} construction from Cash et al. [8]. For our SRC attack, we used the CP-SAT solver from Google's ortools package [46] as our ILP solver.

7 CONCLUSION

We introduce a framework for designing schemes that support range queries over encrypted data in multiple dimensions. In particular, we describe how to turn a broad class of DAG-based spatial range search data structures into an encrypted database that supports range queries. We demonstrate the effectiveness of this framework by developing four schemes that offer trade-offs for space-complexity, query bandwidth, response size, and leakage

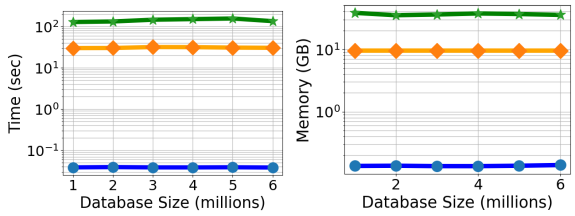


Figure 10: Runtime (left) in seconds and memory requirement (right) in GB of our attacks on the URC (blue circle •), BRC (orange diamond♦) and SRC (green star ⋆) schemes for the Gowalla $[2^5] \times [2^5]$ dataset varying the number of records (in millions).

to suit the needs of a wide variety of applications. We conduct a thorough leakage analysis and present reconstruction attacks.

Our attacks prompt the exploration of mitigation techniques such as frequency smoothing [23], rounding the ranges to a specified integer multiple [42], batching queries, and alternate range decomposition approaches. It would be useful to study database reconstruction attacks on our schemes with weaker assumptions, e.g., attacking the range tree URC/BRC schemes when only having access to a subset of the possible queries and/or with unknown distribution. Another open problem is developing efficient searchable encryption schemes that support other types of spatial queries like aggregate range queries and k-nearest neighbor queries.

ACKNOWLEDGMENTS

Work supported in part by the Kanellakis Fellowship at Brown University and by a gift from the NetApp University Research Fund, a corporate advised fund of Silicon Valley Community Foundation. The authors would also like to thank William Schor for his preliminary contributions to the implementation of the schemes.

REFERENCES

[1] J. L. Bentley and J. H. Friedman. 1979. Data Structures for Range Searching. *ACM Comput. Surv.* 11, 4 (Dec. 1979), 13.

[2] Laura Blackstone, Seny Kamara, and Tarik Moataz. 2020. Revisiting Leakage Abuse Attacks. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society.

[3] D. Bogatov, G. Kollios, and L. Reyzin. 2019. A Comparative Evaluation of Order-Revealing Encryption Schemes and Secure Range-Query Protocols. *Proc. VLDB Endow.* 12, 8 (April 2019), 933–947.

[4] K.S. Booth and G. S. Lueker. 1976. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of computer and system sciences* 13, 3 (1976).

[5] R. Bost. 2016. Sophos: Forward Secure Searchable Encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. New York, NY, USA, 12.

[6] R. Bost, B. Minaud, and O. Ohrimenko. 2017. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. New York, NY, USA, 18.

[7] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-Abuse Attacks Against Searchable Encryption. In *Proc. ACM Conf. on Computer and Communications Security (CCS)*.

[8] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.C. Rosu, and M. Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*.

[9] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.C. Roşu, and M. Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Advances in Cryptology – CRYPTO 2013*. Berlin, Heidelberg.

[10] M. Chase and S. Kamara. 2010. Structured Encryption and Controlled Disclosure. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6477)*.

[11] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. 2011. Friendship and Mobility: User Movement in Location-Based Social Networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Diego, California, USA) *(KDD '11)*. New York, NY, USA, 1082–1090.

[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. 2006. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *Proc. ACM Conf. on Computer and Communications Security*.

[13] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis. 2016. Practical private range search revisited. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD)*.

[14] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, Minos Garofalakis, and Charalampos Papamanthou. 2018. Practical Private Range Search in Depth. *ACM Trans. Database Syst.* 43, 1, Article 2 (2018), 52 pages. https://doi.org/10.1145/3167971

[15] Sabrina De Capitani di Vimercati, Dario Facchinetti, Sara Foresti, Gianluca Oldani, Stefano Paraboschi, Matthew Rossi, and Pierangela Samarati. 2021. Multi-dimensional indexes for point and range queries on outsourced encrypted data. *Proceedings of the GLOBECOM* (2021), 1–1.

[16] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. 2015. Rich Queries on Encrypted Data: Beyond Exact Matches. In *Computer Security – ESORICS 2015*. Cham.

[17] F. Falzon, E. A. Markatou, Akshima, D. Cash, A. Rivkin, J. Stern, and R. Tamassia. 2020. Full Database Reconstruction in Two Dimensions. In *Proc. ACM Conf. on Computer and Communications Security (CCS)*.

[18] R. A. Finkel and J. L. Bentley. 1974. Quad Trees a Data Structure for Retrieval on Composite Keys. 4, 1 (mar 1974), 1–9.

[19] C. Gentry. 2009. *A Fully Homomorphic Encryption Scheme*. Ph.D. Dissertation. Stanford, CA, USA. Advisor(s) Boneh, D.

[20] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption *(CCS '18)*. New York, NY, USA, 18.

[21] O. Goldreich and R. Ostrovsky. 1996. Software Protection and Simulation on Oblivious RAMs. *J. ACM* 43, 3 (May 1996), 43.

[22] Greg Grothaus. [n.d.]. PQTrees. https://github.com/Gregable/pq-trees. Accessed: 2022-01-12.

[23] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. 2020. Pancake: Frequency Smoothing for Encrypted Data Stores. In *USENIX Security Symposium*. 2451–2468.

[24] P. Grubbs, M.S. Lacharité, B. Minaud, and K.G. Paterson. 2018. Pump Up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In *Proc. ACM Conf. on Computer and Communications Security (CCS)*.

[25] P. Grubbs, M. Lacharité, B. Minaud, and K. G. Paterson. 2019. Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks. In *Proc. IEEE Symp. on Security and Privacy (S&P)*.

[26] Z. Gui, O. Johnson, and B. Warinschi. 2019. Encrypted Databases: New Volume Attacks against Range Queries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*.

[27] F. Hahn and F. Kerschbaum. 2016. Poly-Logarithmic Range Queries on Encrypted Data with Small Leakage. In *Proceedings of the 2016 ACM on Cloud Computing Security Workshop* (Vienna, Austria) *(CCSW '16)*. New York, NY, USA, 12.

[28] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society.

[29] S. Kamara and T. Moataz. 2018. SQL on Structurally-Encrypted Databases. In *Advances in Cryptology – ASIACRYPT 2018*. Cham.

[30] Seny Kamara and Tarik Moataz. 2019. Computationally Volume-Hiding Structured Encryption. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11477)*. Springer, 183–213.

[31] S. Kamara and C. Papamanthou. 2013. Parallel and Dynamic Searchable Symmetric Encryption. In *Financial Cryptography and Data Security*. Berlin, Heidelberg.

[32] S. Kamara, C. Papamanthou, and T. Roeder. 2012. Dynamic Searchable Symmetric Encryption. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. New York, NY, USA, 12.

[33] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *Proc. ACM Conf. on Computer and Communications Security 2016 (CCS 2016)*.

[34] Shabnam Kasra Kermanshahi, Shi-Feng Sun, Joseph K. Liu, Ron Steinfeld, Surya Nepal, Wang Fat Lau, and Man Au. 2020. Geometric range search on encrypted data with Forward/Backward security. *IEEE Transactions on Dependable and Secure Computing* (2020), 1–1. https://doi.org/10.1109/TDSC.2020.2982389

[35] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. 2013. Delegatable Pseudorandom Functions and Applications *(CCS '13)*. New York, NY, USA, 669–684.

[36] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. 2020. The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution. In *Proc. IEEE Symp.on Security and Privacy (S&P)*.

[37] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2021. Response-Hiding Encrypted Ranges: Revisiting Security via Parametrized Leakage-Abuse Attacks. In *Proc. IEEE Symp. on Security and Privacy (S&P)*.

[38] M.S. Lacharité, B. Minaud, and K.G. Paterson. 2018. Improved reconstruction attacks on encrypted data using range query leakage. In *Proc. IEEE Symp. on Security and Privacy 2018 (S&P 2018)*.

[39] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. 2005. On Trip Planning Queries in Spatial Databases. In *Advances in Spatial and Temporal Databases*. Berlin, Heidelberg, 273–290.

[40] Evangelia Anna Markatou, Francesca Falzon, Roberto Tamassia, and William Schor. 2021. Reconstructing with Less: Leakage Abuse Attacks in Two Dimensions. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2243–2261.

[41] Evangelia Anna Markatou and Roberto Tamassia. 2019. Full Database Reconstruction with Access and Search Pattern Leakage. In *Proc. Int. Conf on Information Security (ISC)*.

[42] Evangelia Anna Markatou and Roberto Tamassia. 2019. Mitigation Techniques for Attacks on 1-Dimensional Databases that Support Range Queries. In *Information Security - 22nd International Conference, ISC 2019, New York City, NY, USA, September 16-18, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11723)*.

[43] M. Naveed, M. Prabhakaran, and C. A. Gunter. 2014. Dynamic Searchable Encryption via Blind Storage. In *2014 IEEE Symposium on Security and Privacy*.

[44] Simon Oya and Florian Kerschbaum. 2021. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 127–142. https://www.usenix.org/conference/usenixsecurity21/presentation/oya

[45] S. Patel, G. Persiano, K. Yeo, and M. Yung. 2019. Mitigating Leakage in Secure Cloud-Hosted Data Structures: Volume-Hiding for Multi-Maps via Hashing. In *Proc. ACM Conf. on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. New York, NY, USA, 15.

[46] Laurent Perron and Vincent Furnon. 2019. *OR-Tools*. Google. https://developers.google.com/optimization/

[47] David Pouliot and Charles V. Wright. 2016. The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption. In *Proc. ACM Conf. on Computer and Communications Security (CCS)*.

[48] Python Cryptographic Authority. 2018. pyca/cryptography. https://cryptography.io/ version 3.4.7.

[49] P. Rizomiliotis, E. Molla, and S. Gritzalis. 2017. REX: A Searchable Symmetric Encryption Scheme Supporting Range Queries *(CCSW '17)*. New York, NY, USA.

[50] E. Shi, J. Bethencourt, T-H. H. Chan, D. Song, and A. Perrig. 2007. Multi-Dimensional Range Query over Encrypted Data *(SP '07)*. USA, 15.

[51] D. Song, D. Wagner, and A. Perrig. 2000. Practical techniques for searches on encrypted data. In *Proceeding IEEE Symposium on Security and Privacy (S&P)*.

[52] Malte Spitz. 2011. CRAWDAD dataset spitz/cellular (v. 2011-05-04). Downloaded from https://crawdad.org/spitz/cellular/20110504.

[53] B. Wang, Y. Hou, M. Li, H. Wang, and H. Li. 2014. Maple: Scalable Multi-Dimensional Range Search over Encrypted Cloud Data with Tree-Based Index. In *Proc. of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '14)*.

[54] Cong Zuo, Shi-Feng Sun, Joseph K Liu, Jun Shao, and Josef Pieprzyk. 2018. Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security. In *European Symposium on Research in Computer Security (ESORICS) (LNCS)*. Springer, 228–246.

A FORMALIZING LEAKAGE

Structured encryption is parameterized by different leakage functions, which output information about the underlying data structure and its contents. Below, we define two common leakage functions of EMM schemes relevant to this work. Let MM be a multimap with label space $\mathbb{L}$ and volume space $\mathbb{V}$.

- The ***search pattern*** reveals when two queries are equal. It takes as input a multimap MM and a label $\ell \in \mathbb{L}$, and outputs an ID. Without loss of generality we assume a 1-to-1 correspondence between range queries and identifiers: $\mathsf{SP}(\mathsf{MM}, \ell) \mapsto i \in [|\mathbb{L}|]$.
- The ***volume pattern*** of a label ℓ reveals the number of records in $\mathsf{MM}[\ell]$: $\mathsf{Vol}(\mathsf{MM}, \ell) = |\mathsf{MM}[\ell]|$.

In this paper we assume that the underlying EMM scheme is response-hiding, and leaks the multimap size at setup, and the search and volume patterns at query time.

B SECURITY DEFINITION

Definition 7. *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Eval}, \mathsf{Result})$ *be an EMM scheme and let* $\mathcal{L}^\Sigma = (\mathcal{L}^\Sigma_\mathsf{S}, \mathcal{L}^\Sigma_\mathsf{Q})$ *be a tuple of stateful algorithms. For algorithms* $\mathcal{A}$ *and* $\mathcal{S}$*, we two experiments below.*

$\mathbf{Real}^\Sigma_\mathcal{A}(1^\lambda)$

(1) *The adversary* $\mathcal{A}$ *selects a multimap* MM *and gives it to the challenger* $\mathcal{C}$*.*
(2) *The challenger* $\mathcal{C}$ *runs the setup algorithm with* 1^λ *and* MM *as input,* $(K, \mathsf{EMM}) \leftarrow \Sigma.\mathsf{Setup}(\mathsf{MM})$*. The challenger* $\mathcal{C}$ *sends the encrypted multimap* EMM *to the adversary* $\mathcal{A}$*.*
(3) $\mathcal{A}$ *adaptively chooses a polynomial-in-lambda number of labels* $\ell_1, \ldots, \ell_{\mathsf{poly}(\lambda)}$*; for each label* ℓ_i *the adversary sees the token* $t_i \leftarrow \Sigma.\mathsf{Query}(K, \ell_i)$*.*
(4) $\mathcal{A}$ *eventually outputs a bit* $b \in \{0, 1\}$*.*

$\mathbf{Ideal}^\Sigma_{\mathcal{A},\mathcal{S}}(1^\lambda)$

(1) *The adversary* $\mathcal{A}$ *selects a database* D *and gives* $\mathcal{L}^\Sigma_\mathsf{S}(D)$ *to the simulator* $\mathcal{S}$*.*
(2) *The simulator generates an encrypted multimap* EMM *and gives it to the adversary* $\mathcal{A}$*.*
(3) $\mathcal{A}$ *adaptively chooses a polynomial-in-lambda number of labels* $\ell_1, \ldots, \ell_{\mathsf{poly}(\lambda)}$*; for each label* ℓ_i *the challenger computes a pair* $(t_i, \mathsf{st}_\mathcal{S}) \leftarrow \mathcal{L}^\Sigma_\mathsf{Q}(\ell_i, \mathsf{st}_{\mathcal{L}^\Sigma})$ *using the query leakage and the state* $\mathsf{st}_{\mathcal{L}^\Sigma}$ *and gives* t_i *to the adversary* $\mathcal{A}$*.*
(4) $\mathcal{A}$ *eventually outputs a bit* $b \in \{0, 1\}$*.*

Scheme Σ *is* adaptively $\mathcal{L}^\Sigma$-secure *if for all polynomial-time adversaries* $\mathcal{A}$*, there exists a poly-time simulator* $\mathcal{S}$ *such that:*

$$|\Pr[\mathbf{Real}^\Sigma_\mathcal{A}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}^\Sigma_{\mathcal{A},\mathcal{S}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda).$$

C GENERIC FRAMEWORK

C.1 Pseudocode for GenericRS

The pseudocode for GenericRS can be found in Algorithm 11.

C.2 Proof of Theorem 1

Proof. For a contradiction, suppose there is a point $x \in \mathcal{D}$ for which such v does not exist. If $q = x$ is queried, then by Property (v) of Definition 3, RC must return some node that covers q. Algorithm RC must thus return a node $w \in V$ such that $q \subsetneq w.range$. □

C.3 Proof of Theorem 2

Proof. First, we show that BRC must return an exact cover. Consider a point $x \in q$. By assumption (1) there is a leaf node $v \in V$ such that $v.range = x$. Let $p = (s, v_1, \ldots, v_\ell, v)$ be the path from source s to leaf v. By assumption (2) all nodes in T that cover x must be in p.

GenericRS$(1^\lambda, D, \mathcal{D}, \Sigma, (G, \mathsf{RC}))$

Setup$(1^\lambda, D)$:
1 : Initialize empty multimap MM.
2 : **for** node $w \in G$ **do**
3 : MM$[w.range] \leftarrow \{D[x] : x \in w.range\}$
4 : $(K, \mathsf{EMM}) \leftarrow \Sigma.\mathsf{Setup}(1^\lambda, \mathsf{MM})$; **return** (K, EMM)

Query(K, q) :
5 : $W \leftarrow \emptyset$; $\mathbf{t} \leftarrow \emptyset$
6 : $W \leftarrow \mathsf{RC}(G, q)$
7 : **for** $w \in W$ **do** $\mathbf{t} \leftarrow \mathbf{t} \cup \Sigma.\mathsf{Query}(K, w.range)$
8 : permute and **return t**

Eval$(\mathbf{t}, \mathsf{EMM})$:
9 : $\mathbf{c} \leftarrow \emptyset$; **for** $t \in \mathbf{t}$ **do** $\mathbf{c} \leftarrow \mathbf{c} \cup \Sigma.\mathsf{Eval}(t, \mathsf{EMM})$
10 : **return c**

Result$(K, \mathbf{c})$:
11 : $\mathbf{v} \leftarrow \emptyset$; **for** $c \in \mathbf{c}$ **do** $\mathbf{v} \leftarrow \mathbf{v} \cup \Sigma.\mathsf{Result}(K, c)$
12 : **return v**

Figure 11: Algorithms of the generic range encrypted multimap scheme (Definition 2) for a database D with domain $\mathcal{D}$ built from an encrypted multimap scheme Σ (Definition 1) and a range-supporting data structure (G, RC) (Definition 3).

Let u be the most recently explored vertex in p. If u satisfies the if statement on line 4, then u is an exact cover of x and is added to W. Else, we must have that $u.range \cap q \neq \emptyset$ and BRC is called on the children of u. Since this path starts with s then the exploration of p must start, and since p has finite length then this process must also stop. Moreover, since $v.range = x$ then the leaf v satisfies line 4 and x will necessarily be covered without false positives.

Next we show that W is minimal. Suppose not, then there is some other set $W' \subseteq V$ that is minimal. By assumption (2), if two nodes $u, v \in V$ are such that $u.range \subseteq v.range$, then u must be a descendent of v. Combining this observation, with the minimality of W', then there must exist $v' \in W'$ and $v \in W$ such that $v.range \subsetneq v'.range \subseteq q$. But v' must have been explored before v, and lines 4-5 would have added v' to to W, which is a contradiction.

Lastly we prove uniqueness of W. Suppose there exist distinct covers W and W' of minimal size. Then there is a point $x \in q$ in the queried range that is covered by different vertices in the two covers i.e. there exist distinct $v \in W$ and $v' \in W'$ such that without loss of generality $x \subseteq v'.range \subsetneq v.range \subseteq q$. By assumption (2), $v'.range$ forms a non-trivial partition of $v.range$. Thus there exists another node v'' such that $v'.range \cup v''.range \subseteq v.range$ and W' must contain both v' and v'' instead of v, which contradicts the minimality of W'. □

C.4 Proof of Theorem 3

Proof. First we show that SRC returns a cover of q via the following invariant: at the end of every iteration, *cand* is a cover of q. We proceed inductively on the vertices explored. On the first iteration, source s is explored; s satisfies line 4 and thus $cand = s$. Since $s.range = \mathcal{D}$, then *cand* covers q.

Let v be the next vertex explored. If $q \subseteq v.range$ and $|v.range| < |cand.range|$, then by line 8 *cand* is updated to v. Thus *cand* is a cover. Otherwise *cand* is not updated at this iteration, and by our inductive hypothesis, *cand* must cover q.

Next we prove minimality if the number of false positives. Suppose for a contradiction, that SRC returns a cover v' of q that does not minimize the maximum number of possible false positives. Then there exists a vertex $v \in V \setminus \{v'\}$ such that $q \subseteq v.range$ and $|v.range| < |v'.range|$.

If v is explored before v', then when v' is explored we have that $|cand.range| \leq |v.range|$. Since $|v.range| < |v'.range|$, then *cand* would not be updated to v', which is a contradiction.

If v is not explored before v', then v must be explored later otherwise G would not be connected. When v is explored, $|cand.range| \leq |v'.range|$. Since $|v.range| < |v'.range|$, then *cand* must be updated to a vertex whose range query is at least as small as v, which is a contradiction. □

C.5 Proof of Theorem 4

Proof. We construct a stateful simulator $\mathcal{S}$ for Setup and Query.

$(\mathsf{EMM}, \mathrm{st}_\mathcal{S}) \leftarrow \mathcal{S}.\mathsf{SimSetup}(1^\lambda, n, m)$

(1) Invoke the simulator of the underlying EMM scheme to initialize an encrypted multimap EMM with n random values.
(2) Return EMM.

$(\mathsf{EMM}, \mathrm{st}_\mathcal{S}) \leftarrow \mathcal{S}.\mathsf{SimQuery}(1^\lambda, (\mathsf{Str}(D, q^{(i)}))_{i \in [t]})$

(1) For each $\mathsf{SP}(\mathsf{MM}, v.range)$ such that $v \in \mathsf{RC}(G, s, q^{(t)})$:
   (a) Determine if $v.rage$ has been queried before using the search patterns $\mathsf{SP}(\mathsf{MM}, w.range)$ for all nodes in the covers $w \in \mathsf{RC}(G, s, q^{(i)})$ such that $i \in [t-1]$.
   (b) If yes, then simulator $\mathcal{S}$ uses state $\mathrm{st}_\mathcal{S}$ to return the same result as before.
   (c) Else simulator $\mathcal{S}$ uses the RC algorithm and invokes the simulator of the EMM scheme Σ on $\mathsf{SP}(\mathsf{MM}, v.range)$, obtains the result for $v.range$, and updates its state $\mathrm{st}_\mathcal{S}$.

Note that the simulator uses SP, RC, and G to correctly simulate the results of the structural leakage.

It remains to show that for all probabilistic poly-time adversaries $\mathcal{A}$, the probability $|\Pr[\mathbf{Real}_\mathcal{A}^{\mathsf{GenericRS}} = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathsf{GenericRS}} = 1]|$ is negligibly small. We define the following three games and conclude with a hybrid argument.

Hyb0: This is identical to $\mathbf{Real}_\mathcal{A}^{\mathsf{GenericRS}}$.

Hyb1: This is identical to **Hyb0**, except that every encrypted record c in EMM is replaced with a random string r of correct length. Note that each record replacement is done via a single hybrid, and we thus combine a total of $f(n, m)$ hybrids into this single step.

Hyb2: This is identical to **Hyb1**, except that instead of invoking Σ.Setup and Σ.Query we invoke the simulator of the underlying EMM scheme.

$|\Pr[\mathbf{Hyb0}] - \Pr[\mathbf{Hyb1}]|$ is negligibly small, otherwise the CPA security of the underlying symmetric encryption scheme would be broken with non-negligible probability. Similarly, $|\Pr[\mathbf{Hyb1}] - \Pr[\mathbf{Hyb2}]|$ is negligibly small, otherwise the security of the underlying EMM scheme would be broken with non-negligible probability. Since the distribution of **Hyb2** is identical to $\mathbf{Ideal}_\mathcal{A}^{\mathsf{GenericRS}}$ this concludes our proof. □

D SCHEMES

D.1 Linear Scheme

D.1.1 Pseudocode for LRC.

Algorithm 9: $\mathsf{LRC}(T, q)$

$W \leftarrow \emptyset$
for $x \subseteq q$ **do** $W \leftarrow W \cup \{V[x]\}$
return W

D.1.2 Complexity of the Linear Scheme.

Theorem 17. *Let G_L be the star DAG for a database D of size n on a d-dimensional domain $\mathcal{D}$ of size m, and let* LRC *be the linear covering algorithm defined in Algorithm 9. We have that (G_{RT}, LRC) is a range-supporting data structure (Definition 3) and the range encrypted multimap scheme derived from it (Definition 5) uses space $O(n+m)$. Also, a query with range size R and result size r has query size R and response size r.*

Proof. It is straightforward to verify properties (i) to (iv). It remains to show that algorithm LRC runs in polynomial-time when invoked on the source of G_L. LRC is implemented such that the sinks of G_L are stored in an array, $V[\mathcal{D}]$, indexed by domain point. Let q be any range query on the domain. Initializing an empty set W can be done in constant time. Looping through $x \in q$ and adding $V[x]$ to W takes time $O(R)$, where R is the size of the range q.

We now prove the complexity. The linear scheme generates a multimap with m labels, one for each sink in G_L. Each record is stored once with its corresponding point value. The index has size $n+m$. When the client issues a range query of size R, the client computes R search tokens and sends them to the server. Each search corresponds to the domain points and each record is stored once, hence a it has a response size of $O(r)$. □

D.2 Range tree Scheme

D.2.1 Proof of Theorem 5. We first restate the one-dimensional analogue proved in [35].

Theorem 18 ([35]). *Let G_{RT} be a one-dimensional range tree on domain $[m]$. Given range queries q and q' of the same size R, let W and W' be their respective covers returned by algorithm* URC *on G_{RT}. We have $|W| = |W'|$.*

We now give the proof of Theorem 5.

Proof. We prove the following invariant: at the end of the i^{th} iteration of the for loop on line 4, the set W is the minimal cover of the range $q_1 \times \cdots \times q_i \times [m_{i+1}] \times \cdots \times [m_d]$, where $q = q_1 \times \cdots \times q_d$. We proceed by induction on i.

Let $i = 1$. At the start of the first iteration $W = \{s\}$. Let $\widehat{T} \subseteq T$ be the subtree on $[m_1]$ rooted at s. Note that the canonical ranges of $\widehat{T}$ are comprised of dyadic ranges in dimension 1 and the whole domain in dimensions 2 to d (Equation 1). On line 8 we thus construct a query $\hat{q} = q_1 \times [m_2] \times \cdots \times [m_d]$. Since the first dimension can be covered by dyadic ranges along $[m_1]$, then by Corollary 1, W contains a minimal and unique cover of $\hat{q}$ in $\widehat{T}$.

Now let $i > 1$ and w be any vertex in W. At the start of the ith iteration, W must comprise of the unique, minimal cover of $q' = q_1 \times \cdots \times q_{i-1} \times [m_i] \times \cdots \times [m_d]$. In particular, $q' = \bigcup_{w \in W} w.range$. Let $\widehat{T} \subseteq T$ be the subtree on $[m_i]$ rooted at w, and let $w_1 \times \cdots \times w_d$ be the canonical range of w. On line 8 we construct a query $\hat{q} = w_1 \times \cdots \times w_{i-1} \times q_i \times [m_{i+1}] \times \cdots \times [m_d]$. Note that $\hat{q} \subseteq w.range$ and $\hat{q}$ is the union of canonical ranges of the leaves of $\hat{T}$. By Corollary 1, Algorithm 3 (BRC_{RT}) returns the minimal and unique cover of $\hat{q}$ in $\widehat{T}$. This argument holds for all vetices $w \in W$.

If the resulting cover is not unique, then there must be another distinct cover of q' but that would either contradict the inductive hypothesis or Corollary 1. A similar argument can be made for minimality. The inductive hypothesis holds at the end of the ith iteration, which concludes our proof. □

D.2.2 Proof of Theorem 6.

Proof. Let $\widehat{T}$ be one of the binary subtrees of T on $[m_i]$ for any $i \in [d]$ with a source w. By construction, the nodes of $\widehat{T}$ partition $w.range$ with respect to the dyadic ranges of $[m_i]$. We now introduce the following useful lemma.

Lemma 1. *Let $i \in [d]$ and W be the set of range covers at the end of iteration $i-1$. For $w \in W$, let $\hat{q}$ and $\widehat{T}$ be defined as in the equivalent of Algorithm 3 (URC_{RT}), respectively. Then the resulting cover W' has the same size for all $w \in W$.*

Proof. For each $w \in W$, its corresponding range query $\hat{q}$ has the same size range along the ith dimension i.e. size $|q_i|$. Since we are applying URC to a single binary tree over one dimension, then by Theorem 18, W' will have the same resulting size. □

More generally, note the following. Let $|q_i|$ be fixed, and let $\widehat{T}$ be a binary tree over $[m]$ for any integer m that's a power of 2. Then the size of a decomposition of a range query of size $|q_i|$ in $\widehat{T}$ must have the same size.

Now let q be a range query of size $R = R_1 \times \cdots \times R_d$ and let q' be a range query of size $R = R_{\sigma(1)} \times \cdots \times R_{\sigma(d)}$. For each R_i let C_i be the size of the resulting cover W' when a universal range cover of q_i is computed on any binary tree on some domain $[m]$. Such C_i must exist by the above observation. Then when we apply $\mathsf{URC}_{RT}(T, q, s)$, the resulting cover has size $C_1 \times C_2 \times \cdots \times C_d$.

Applying Lemma 1, we see that when we compute $\mathsf{URC}_{RT}(T, q, s)$, the resulting cover has size $C_{\sigma(1)} \times C_{\sigma(2)} \times \cdots \times C_{\sigma(d)}$. Thus we can conclude that q and q' have the same size URC_{RT} covers. □

D.2.3 Complexity of the Range tree Scheme.

Theorem 19. *Let G_{RT} be the range tree for a database D of size n on a d-dimensional domain $\mathcal{D}$ of size m, and let BRC_{RT} and URC_{RT} be the range covering algorithms defined in Section 4.2. We have that $(G_{RT}, \mathsf{BRC}_{RT})$ and $(G_{RT}, \mathsf{URC}_{RT})$ are range-supporting data structures (Definition 3) and the range encrypted multimap schemes derived from them (Definition 5) use space $O(n + m \log^d m)$. Also, a query with range size R and result size r has query size $O(\log^d R)$ and response size r.*

Proof. We give the proof for the range encrypted multimap scheme built from $(G_{RT}, \mathsf{BRC}_{RT})$. It is straightforward to check that DAG G_{RT} satisfies properties (i) to (iv) of Definition 3. We now show that BRC_{RT} runs in poly-time when called on G_{RT}.

At the start of the i^{th} iteration of the for loop on line 4, W contains at most $\prod_{j=1}^{i-1} \log R_j$ nodes where $R = R_1 \times \cdots \times R_d$ is the size of the queried range. Parsing $w.range$ and computing $\hat{q}$ can be done in constant time. Computing the subtree rooted at w takes time linear in the number of nodes of $\widehat{T} \subseteq G_{RT}$. Next, observe that

BRC does a depth-first search traversal of the vertices of the input tree, thus this subroutine takes time linear in the number of nodes of $\widehat{T}$. When $m_i = O(m)$, the outer for loop takes time $O(m \log^d m)$ and therefore BRC_{RT} runs in polynomial time.

We now prove the complexity. The range tree has $\prod_{i=1}^{d} 2m_i$ nodes that each correspond to a label in the multimap; each record is stored $\prod_{i=1}^{d} \log m_i$ times for a total storage of $(m + n \log^d m)$. Algorithm BRC guarantees that any range can be covered by a logarithmic number of pre-computed ranges [13] in a range tree; Since we apply BRC once for every every dimension, any range over $\mathcal{D}$ can be covered with $O(\log^d R)$ canonical ranges. Since BRC_{RT} iteratively applies BRC on one-dimensional range trees, then applying Theorem 2, the cover is minimal and has no false positives. Moreover, the cover returned by BRC on a one dimensional range tree is disjoint, and hence the cover returned by BRC_{RT} is also disjoint. Thus the response size is $O(r)$.

The proof for the range encrypted multimap scheme built from $(G_{RT}, \mathsf{URC}_{RT})$ follows a similar argument and is thus omitted. □

D.3 QDAG SRC Scheme

D.3.1 False Positives.

Lemma 2. *Given QDAG $G_{QS} = (V, E)$ over domain $\mathcal{D} = [m_1] \times \cdots \times [m_d]$ and any range q in $\mathcal{D}$ of size $R = R_1 \times \cdots \times R_d$, there exists a vertex $v \in V$ such that $q \subseteq v.range$ and $v.range$ has size $O(R^d)$.*

Proof. Recall that the number of nodes in a quadtree scheme is $O(m)$. In the quadtree, at the j-th level, each of the d dimensions is partitioned into 2^{n-j} axis-aligned segments. Thus, at the j-th level of the quadtree we have partitioned the domain into $2^{(n-j)d}$ hypercubes.

In the QDAG, we shift each hypercube by the legnth of half of the edge of the hypercube along each dimension. Thus at the j-th level of the QDAG we have at most $2^d 2^{(n-j)d} = O(2^{(n-j)d})$ hypercubes. Each hypercube corresponds to a node in the QDAG, so size of the QDAG is upper bounded by a constant factor of 2^d times the size of the region quadtree. □

D.3.2 Space Usage.

Lemma 3. *Let D be a database with n records and a domain $\mathcal{D}$ of size m. Then the size of the QDAG on $\mathcal{D}$ is $O(m)$.*

Proof. Let q be any range query of size R. We will show that this range can be covered by a vertex $v \in V$ such that $v.range$ has size $O(R^d)$. First note, there exists some minimal integer j such that for all i, $R_i \leq 2^j \leq 2R_i$.

Case 1: range query q is covered by a vertex $v \in V$ such that $v.range$ has size 2^{jd}. Thus, q is covered by a range of size

$$O(\max\{R_1, \ldots, R_d\}^d) = O(R^d).$$

Case 2: range query q is *not* covered by a vertex $v \in V$ such that $v.range$ has size 2^{jd}. Along each dimension, the range q must intersect with at most 2 distinct ranges $v.range, v'.range$ each of size 2^{jd} where $v, v' \in V$. Since ranges of the same size are shifted by lengths of 2^{j-1} there must exist a hypercube in $\mathcal{D}$ with edges of length $2^j + 2^{j-1}$ that completely contains q. Note this hypercube does not correspond to a vertex of G_{QS}.

Let $q' = [a_1, b_1] \times \cdots \times [a_d, b_d]$ define this hypercube. Consider an edge $[a_i, b_i]$ of this hypercube. For each $i \in [d]$ there is a set of vertices $V_i \subset V$ such that for all $v \in V_i$, $v.range$ is of size $2^{(j+1)d}$ and it covers $[a_i, b_i]$. In the i-th dimension $v.range$ must either start at a_i or at $a_i - 2^j$. Since the hypercubes of size $2^{(j+1)d}$ are tiling the entire domain with shifts of 2^j along each dimension we can thus find a vertex $v^* \in T$ that contains q' and thus also contains q. The range $v^*.range$ has size $2^{(j+1)d} = O(2^{jd}) = O(R^d)$. □

D.3.3 Complexity.

Theorem 20. *Let G_{QS} be the QDAG for a database D of size n on a d-dimensional domain $\mathcal{D}$ of size m, and let* SRC *be the linear covering algorithm defined in Algorithm 2. We have that (G_{QS}, SRC) is a range-supporting data structure (Definition 3) and the range encrypted multimap scheme derived from it (Definition 5) uses space $O(m + n \log m)$. Also, a query with range size R and result size r has query size 1 and response size $O(r + R^2)$.*

Proof. QDAGs satisfy properties (i) to (iv) of Definition 3. We now show that the range cover algorithm SRC runs in poly-time when called on G_{QS} and its source node s.

SRC explores the graph in a depth-first search manner and each node is visited at most once. If $q \subseteq v.range$, then *cand* is updated to v. Then for every vertex w such that $(v, w) \in E$, SRC is recursively called on G_{QS} and w, and a new vertex t is obtained. If $|t.range| < |cand.range|$, then *cand* is updated to t. By Lemma 3, there are $O(m)$ nodes in G_{QS}, so visiting every node once takes time $O(m)$; updating *cand* takes constant time.

We now prove the complexity. By Lemma 3, the number of nodes in G_{QS} is a constant factor larger than the corresponding quadtree and thus the corresponding range encrypted multimap scheme has an asymptotic storage complexity of $O(m + n \log m)$. To query this scheme, the client generates a single search token, and thus the scheme has a query complexity of 1. By Lemma 2, the total number of false positives for any given query is $O(R^d)$, where R is the size of the query issued. Thus, the total response size is $O(r + R^2)$. □

E LEAKAGE ANALYSIS

E.1 Proof of Theorem 8

Proof. It is straight forward to see how one can build VM and FM from the multiset $\{\{\mathrm{Str}(D, q^i))\}\}_{i \in [\ell]}$. To show the reverse, we will construct the structure pattern using VM and FM.

Initialize an empty multiset S. Then for each tokenset $\mathbf{t}^{(i)}$ in VM:

(1) Initialize an empty dictionary M.
(2) For each $t \in \mathbf{t}^{(i)}$ set $M[t] \leftarrow \mathsf{VM}[t]$.
(3) $f \leftarrow \mathsf{FM}[\mathbf{t}^{(i)}]$
(4) Add f copies of M to the multiset S.

Each issued query $q^{(i)}$ corresponds to a tokenset $\mathbf{t}^{(i)}$ i.e. the search pattern of the canonical ranges that cover q. VM associates each tokenset with the observed volume i.e. the volume pattern of the response. Since each map M in S is added as many times as the corresponding tokenset has been observed, then the structure pattern multiset is in one-to-one correspondence with the multiset S. □

E.2 Proof of Theorem 9

Proof. Consider a database D encrypted with the linear scheme. Consider a token t and its neighboring tokens t_0, t_0', t_1, t_1'.... Two tokens are neighboring if they correspond to the same values in all dimensions but one and in the remaining dimension their value differs by 1. There exists a range query that issues t with each one of its neighboring search tokens. There exists no query of size two with t that does not contain one of its neighbors as that query would not correspond to a valid range. Combining all these queries, can construct a grid that covers the entire domain. This grid is dense and does not allow for reflectable components as in [17]. Thus the reconstruction space only includes transformations of D corresponding to the symmetries of a d-cube. □

E.3 Proof of Theorem 10

Proof. The first step of Algorithm 4 is to find any queries that correspond to a set of search tokens of prime size, say set Q. We know that the size of the range being queried is leaked, as it is the number of search tokens the client sends the server. If a range has prime size p, then the query covers p points in one dimension and one point in the remaining dimensions. Thus, all queries in Q query are one-dimensional sections. The next step is to group queries that come from the same one-dimensional section. Note that if two queries' search token intersection contains two or more elements, then the queries must correspond to ranges along the same one-dimensional section. We thus group queries in their corresponding one-dimensional section, and create a PQ-tree for each one-dimensional section. The attack then generates a graph G that contains an edge between neighboring search tokens, representing a partial order reconstruction of the search tokens. Once we map the search tokens to their corresponding volumes, we achieve partial database reconstruction. If the adversary has observed enough queries for the order of the individual one-dimensional sections to be fully reconstructed, graph G is a d-dimensional grid fully ordering all search tokens, and thus achieving full database reconstruction.

To achieve FDR, every PQ-tree must have enough information to reconstruct the order of each one-dimensional section. Consider a one-dimensional section, e.g. a row R. The search tokens in R share all values but one, the one corresponding to the first dimension. Thus, their values span from 1 to m_1. Split the search tokens in two groups: A includes all search tokens with values less than $m_1/2$ in the first dimension and B contains the remaining points in R. The PQ-tree can order these search tokens if in its input there exists a range that starts before and a range that starts after every search token. Thus, if the PQ-tree observes a range that starts before every point in A or ends before every point in B or after the last point of B, it can fully order the search tokens in R. Let's count the number of range queries that start at a specific point in A, end anywhere in B and have prime length. There are more small prime numbers than larger. Thus, the worst case scenario is our starting point being in the beginning of A. Thus, the possible size of our range is between $m_1/2$ and m_1. We approximate the number of prime numbers between $m_1/2$ and N_1 to be around $\frac{m_1}{\log m_1} - \frac{m_1/2}{\log m_1/2} > \frac{m_1/6}{\log m_1/6}$, for $m_1 > 26$. Thus, the probability that a range query satisfies these constraints is $\frac{1}{m_1 6 \log m_1/6m_2^2...m_d^2}$. Let $x = m_1 6 \log m_1/6m_2^2...m_d^2$. After observing $10x \log x$ queries, then we will not have observed even one query satisfying the constraints with probability $(1-1/x)^{10x\log x} \approx \frac{1}{x^{10}}$. There are $m_1/2$ such queries from A and $m_1/2$ similar such queries from B. TBy union bound, the probability that even one of them is missing is approximately $\frac{m_1}{(m_1 6 \log m_2/6m_3^2...m_d^2)^{10}} \leq \frac{1}{m^5}$. There are m_0 rows, thus the probability we missed one of them is $\leq \frac{m_1}{m^5}$. We can make a similar argument for each PQ tree, concluding that if the adversary observes $\sum_{i=1}^{d} \Omega\left(\frac{m^2}{m_i} \log m_i \cdot \log\left(\frac{m^2}{m_i} \log m_i\right)\right)$ queries, Algorithm 4 achieves FDR with probability greater than $1 - \frac{1}{m^2}$.

There are $O(m^2)$ observed search tokens and responses, each of which require $O(m)$ space. The algorithm thus requires $O(m^3)$ storage. Algorithm 4 first identifies all tokensets of prime size which takes $O(m^3)$ time. Then, the Algorithm identifies which tokens correspond to the same one-dimensional slice. This requires a loop over all tokensets $O(m^2)$ and on each loop doing a set intersection between sets of size $O(m)$, $O(m^2)$ times. Thus, it takes $O(m^5)$ time. We then construct a PQ tree for each one-dimensional slice and create the augmented graph G, which takes $O(m^2)$ time. Thus, Algorithm 4 takes $O(m^5)$ time, $O(m^3)$ space and succeeds with probability greater than $1 - \frac{1}{m^2}$ after observing $\sum_{i=1}^{d} \Omega\left(\frac{m^2}{m_i} \log m_i \cdot \log\left(\frac{m^2}{m_i} \log m_i\right)\right)$ queries uniformly at random. □

E.4 Proof of Theorem 11

Proof. Let st be a search token corresponding to a point query $(i_1, i_2, \ldots, i_d)$ and st_i, st_i' be its neighboring search tokens in dimension i. Note that if the client wishes to perform a point query, they send exactly one search token to the server. The only other times they send only one search token are when they query either one point or the whole range in all dimensions. Let Q be a set with all such search tokens.

If the client queries a neighboring value of $(i_1, i_2, \ldots, i_d)$, i.e. a value that differs in only one dimension by 1, they have to send the two neighboring search tokens. Because the client picked URC, they cannot send only one search token, even if one exists that spans that range. Thus, given all range queries corresponding to two search tokens, they can construct a grid spanning the domain of the database fully ordering the point search tokens, and recovering their corresponding value. This leakage does not allow for reflectable components like [17], and the reconstruction space corresponds to the symmetries of a d-cube. □

E.5 Proof of Theorem 12

Proof. The first step of Algorithm 5 is to find queries that correspond to exactly one token, and place them in a set Q_1. Since the client is using URC, the chosen sub-queries are all combinations of the client's one-dimensional range choices. Thus, the number of search tokens that corresponds to a range query is the product of the number of sub-queries in each dimension. Since the product is one, we conclude that in each dimension, the client is querying either one value or everything.

The algorithm builds a graph, G, that has an edge between two search tokens in Q_1 if there is a query that corresponds to that pair of search tokens. The largest component of G corresponds to a partial order reconstruction of the search tokens. We now show that (i) there can be an edge in G between any two neighboring point-value tokens; (ii) no edge exists between point-value tokens that are not neighbors; and (iii) the largest component of the graph contains the point-value tokens.

(i) Because the client uses URC, any queries of size two must correspond to two search tokens. Thus, a range query of two point values must correspond to two neighboring search tokens. (ii) For two search tokens to be in a query together they must form a valid hyperrectangle. A range corresponding to a single point can only form a valid rectangle when combined with another neighboring single point. All other options in Q_1 are already multiple-dimensional rectangles, which combined with a single point do not make a valid range. Note that this is not true in general, however members of Q_1 either query a single point on a given dimension or all of them. It could be possible to generate a valid range by adding a single point to the end of a line segment, however in this case the line segment spans the whole range. (iii) Only members of Q_1 corresponding to ranges of the same dimension can have an edge between them in G. There are more point-valued ranges than valid ranges in all other cases. Thus, the largest component will be the one corresponding to point queries.

Since volume is leaked, volumes can be associated with their corresponding tokens. The graph G has size $O(m)$. It takes $O(m^2)$ time to look for all queries of size one and two, and $O(m^2)$ to construct G. In case the adversary has observed all possible queries, which happens with high probability after $\Omega(m^2 \log m)$ uniformly distributed queries by the coupon collector principle, graph G contains all relevant search tokens and corresponding edges. Since it takes at least $O(m^2 \log^d m)$ to read the search tokens and their responses, our algorithm requires $O(m^2 \log^d m)$ time. The algorithm requires $O(m^2 \log^d m)$ space to store the search tokens and their responses. □

E.6 Proof of Theorem 14

Proof. First, we show that we can reconstruct the inner grid database $\mathcal{D}$ and then we show that we can reconstruct the volumes of the extreme points as well (up to the symmetries of the square). Finally, we show that our algorithm succeeds with probability greater than $1 - \frac{1}{m^2}$ after observing $\Omega(m^2 \log m)$ queries uniformly random queries in $O(m^4)$ time.

The first step of the algorithm is to construct a co-occurrence graph G with nodes search tokens (i.e. range tree nodes), and an edge between two nodes if there is a tokenset consisting of the two of them. Then, for each tokenset S of size greater than two, we observe hyperrectangle (Lemma 4) graph G_S of G induced by the nodes of S. In a one-dimensional query, this is a line graph. G_S is also a line graph when the node of G_S cover the same ranges in all dimensions but one. In case of a line graph, G_S only has two nodes that have the smallest degree. In all other cases, there are four or more nodes that have the smallest degree, as they are the corners of the hyperrectangle.

We are able to identify all one-dimensional queries (including some higher dimensional ones) by noting which tokensets have only two nodes with the smallest degree in G_S. The next step is to identify the inner nodes that cover two domain points. We measure the *edgecount* of each edge $e = (s_1, s_2)$: the number of times s_1 and s_2 appear in our identified queries together. Note that a token corresponding to a query of size 2, s_1, (e.g. gh in Figure 4) is connected in G with another token s_2, such that their edgecount is exactly 2. There are only two possible tokensets that contain s_1 and s_2 together, (s_1, s_2) and (s_1, s_2, s_3), for some token s_3. Extending the range in either direction will replace either s_1 or s_2 with their ancestors. For example, there are only 2 tokensets containing gh and i together,(gh, i) and (gh, i, f). Extending the range in one direction replaces gh with $\{efgh\}$ and in the other direction replaces i with $\{ij\}$. Note that any other one-dimensional tokens that cover a larger range, have edges with higher edge counts as there are more possible tokensets. For example, there are three tokensets containing $efgh$ and i together,$(efgh, i)$, $(efgh, i, d)$ and $(efgh, i, cd)$. Extending the range in one direction replaces gh with $\{efgh\}$ and in the other direction replaces i with $\{ij\}$. It is possible that inner nodes that cover multi-dimensional ranges have *edgecount* of 2 with a non-inner node. However, these tokens cannot be connected to leaf tokens as they would not form valid ranges, and thus such edges do not exist in G. In the end, we only consider the largest component of the graph, which contains the leaf tokens, ignoring other smaller components that may contain these multi-dimensional tokens.

At this point, we have identified the inner nodes that cover pairs of points and most of the point-query tokens. We still have to distinguish between certain leaf tokens and certain boundary tokens. Specifically, some inner tokens have isomorphic edges with edgecount of two. One edge connects to a boundary node, and the other edge connects to an inner node. For example, in Figure 4, nodes d and $abcd$ are isomorphic. We are able to extract the correct token, by picking the one with the most edges in the original co-occurence matrix. The leaf node will always have one more edge than the boundary node, as the leaf node can be in a tokenset with the boundary node's sibling.

Now, we have identified all the pair token nodes and the tokens (or volumes) of all non-extreme leaf nodes. However, due to the nature of BRC and graph G, they are not in order. By doing a series of swaps and contractions, removing the pair-token nodes and putting the leaf nodes in order, we can reconstruct the inner grid of the database.

Once we have reconstructed the inner grid of the database, we can now reconstruct the volumes at the extreme points. Note that for the inner grid, we were able to identify which search token corresponds to which domain point. However, we cannot do the same for the extreme points, we can only extrapolate the volumes. The reason is that due to BRC, the search tokens for extreme points of the database often appear alone. For example, in Figures 4 and 5, the corner search tokens appear identical in the co-occurrence graph (nodes a, p and a, d, p, m respectively).

In our d-dimensional grid, in place of each domain point p extreme in one dimension is a search token s covering p and its neighbor p_n, where s has exactly one neighbor in our graph G', the search token covering p_n. We can thus extrapolate the volume of p from the volumes of s and p_n. We are unable to generalize this

technique to tokens extreme in multiple dimensions as there is no such token.

Let's assume we have extrapolated all volumes of domain points extreme in up to $i-1$ dimensions. For each domain value v (extreme in i dimensions) we have yet to extrapolate, we have to find a basic i-dimensional cube c with sides of size 2, that contains v. In order to identify this cube, we leverage the structure of BRC tokensets. The range query that covers an i-dimensional cube of side length 3, c_3, which includes v, consists of a tokenset of size i+2. These tokens correspond to an i-dimensional cube of side length 2, c_2, (containing v), one token corresponding to a point value t diagonal to v right outside c_2, and i 2×1 rectangles, set R. Essentially, the rectangles extend c_2 by one value in each dimension. We need one more point to cover all of c_3, which is t. Using the co-occurrence graph and the inner grid, we are able to identify tokens from R and t. Then, the smallest response that contains all of them, must contain c_2, in order to form a valid range. Then, using the volume of c_2, we can extrapolate the volume of v. For example in Figure 5, to extrapolate the volume of a, the 2-dimensional cube of side length 2 is $\{abef\}$, the 2 2×1 rectangles are $\{ij\}$ and $\{cg\}$ and $t = k$. This way, we can extrapolate the volume of domain points extreme in i dimensions. Thus, by induction, our algorithm can find the volumes of all extreme domain points.

The first step of the algorithm is to create dictionaries E, Q, which takes $O(m^2 \log^d m)$, as we have to go through all possible queries and their responses. Then, we go through $O(m^2)$ tokensets and for each construct a graph with their tokens and remove relevant edges from G. This takes $O(m^4)$ time. Then, we disambiguate identical components of the graph and contract edges of the graph, which takes at most $O(m^2)$ time. Then, we swap one-dimensional section of the graph, which takes $O(m^2)$ time.

We then have to identify volumes of the extreme points. There are $O(m)$ such points, and for each point we find a constant number of neighbors in two graphs, which takes $O(m^2)$ time. Then, we look through the tokensets to identify the required tokenset and extract the extreme point's volume. Finding the extreme point values takes $O(m^3)$ time. Thus, in total, our attack takes $O(m^4)$ time.

Our attack needs to observe all possible queries and their responses to work. Using a coupon collector argument, we observe all possible queries after the $\Omega(m^2 \log m)$ queries have been issued under a uniform distribution with probability greater than $1 - \frac{1}{m^2}$.

The adversary needs $O(m^2 \log^d m)$ space to store the search tokens and their responses. Graph G requires $O(m^2)$ space, with $O(m)$ nodes and $O(m^2)$ edges, similar to the temporary storage required by the G_S's. Most work on the algorithm is done on these graphs and there is a constant number of additional data-structures used, all requiring less than $O(m^2)$ space. Thus, the algorithm requires $O(m^2 \log^d m)$ space.

□

Lemma 4. *Let D be a d-dimensional database, over domain $\mathcal{D} = [m_1] \times \ldots \times [m_d]$, with $m = m_1 \cdot \ldots \cdot m_d$, which is encrypted under the range tree scheme and leaks volume, search pattern and sub-queries under* BRC. *Let G be the co-occurrence graph with nodes the tokens of the range tree, and an edge between two nodes if they compose a tokenset. Graph G_S induced by the nodes of a tokenset S on G forms a hyperrectangle.*

Proof. We prove this lemma by induction on the number of dimensions of the range that corresponds to tokenset S. In case of a 1D range query (on range r), we show that the corresponding tokenset S forms a line graph G_S of G induced on the nodes of S. A token $s \in S$ has one or two edges in G_S (unless $|S| = 1$).

Let us say that s_1 covers one of the endpoints of r, range r_1. As S forms a valid range, some token must cover the domain point $p \in r$, right next to r_1. Let us say token s_2 covers r_2, with $p \in r_2$. As $r_1 + r_2$ form a valid range, there is an edge between s_1 and s_2 in G_S. Notably, s_1 cannot be connected to any other token in S, as it would not form a valid range. On the other hand, $\exists p' \in r$ right outside r_2 (if $|S| > 1$). Thus, some token $s_3 \in S$ must cover it (with range r_3). Since $r_2 + r_3$ forms a valid range there exists an edge between s_2 and s_3. Since this is a 1D query, s_2 cannot have any other connections. Similarly we can show that any token of S that does not correspond to range covering an endpoint of r has two edges and the remaining two have one edge. Thus, G_S is a line graph.

Suppose that any tokenset covering an $(i-1)$-dimensional range query forms a hyperrectangle. Let S cover an i-dimensional range r. Let the last dimension of r be of size k. There are k $(i-1)$-dimensional ranges $r_j, j \in [1, k]$ that can be combined to cover r. Let r_j and r_{j+1} differ by one in the last dimension. Let tokens s_j (covering some part of r_j) and s_{j+1} (covering some part of r_{j+1}) cover the same ranges in the first $i-1$ dimensions. The combination of the ranges they cover is valid. Thus, there either exists an edge between them in G or there exists a token that covers their ranges combined.

Let T_i be the set of tokensets S_j covering each of the k $(i-1)$-dimensional ranges r_j. For each neighboring pair of tokensets in T_i, there either exists new tokenset S' that covers their combined ranges (larger tokens apply) or the tokens are included in the BRC response (if no possible combination with neighbors exists). Note that it cannot be the case that only a subset of the neighboring tokensets can be combined, as the tokens are created in the same way for the different neighboring ranges. In case they are included in the BRC response, this pair of tokensets forms a hyper-rectangle (with their edges from G). Additionally, any new tokenset (replacing a previous pair) must also form a hyper-rectangle with its neighbors. Since all neighboring tokensets form hyper-rectangles, all of them together in G_S form a hyperrectangle of dimension up to i, (potentially the hyper-rectangles $(i-1)$-dimensional ranges form hyperrectangles of dimension smaller than $i-1$).

□

E.7 Explanation of SRC Lower Bound

Before demonstrating a lowerbound for the size of the reconstruction space of the QDAG with SRC, we first build intuition with an example of a quadtree with SRC. Recall that a *region quadtree* is a tree that recursively partitions a square domain into 2^2 quadrants. Each non-leaf node v has four children; each child of v is associated with a quadrant of $v.range$. Using SRC as a covering algorithm for a quadtree results in a false positive rate of $O(m)$. As we will soon see, the quadtree SRC scheme – in contrast to the QDAG SRC scheme – is more secure at the expense of significantly more false positives.

We now demonstrate an assignment of volumes to the database resulting in a reconstruction space exponential in m. Note that each

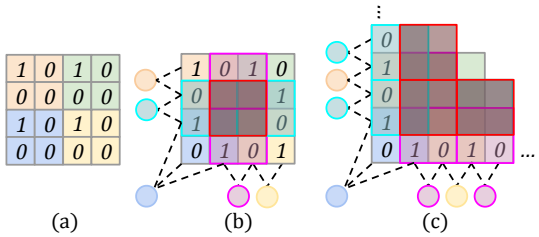


Figure 12: Here we demonstrate an assignment of volumes that gives a lowerbound on the reconstruction space of (a) the quadtree with SRC and (b)-(c) the QDAG with SRC.

leaf node has 3 siblings. For each set of 4 sibling leaf nodes assign a volume of 1 to one leaf and 0 to the other siblings. Each leaf has a frequency of one and thus any set of volumes corresponding to the four siblings can be permuted; there are 2^2 unique permutations per set of 4 siblings, and $m/(2^2)$ such sets of siblings. More generally, we see that the reconstruction space of the quadtree is lowerbounded by $(2^d)^{m/2^d} >> 2^d(d!)$.

In contrast, the QDAG with the SRC range covering algorithm, offers a smaller false positive rate at the expense of a significantly smaller reconstruction space; we note however that the reconstruction is still $O(2^{d-1})$ greater than the symmetries of the hypercube. This is because the additional nodes and edges in the QDAG create a number of additional restrictions that the volume assignments must satisfy hence reducing the total number of possible symmetries when compared to the region quadtree. In order to maintain the same volume assignments to the leaf nodes' parents, we cannot independently permute the volumes of the leaves.

Each QDAG node corresponding to each domain point not at the edge of the database is covered by an additional three nodes (compared to the quadtree). Assign each such domain point a unique value greater than 1. Each QDAG node corresponding to the domain points at the edge of the database is covered by 0 additional nodes (in the case of a corner) or 1 additional node otherwise. Assign each of these external domain points alternating bit volumes (See Figure 12). Since there is an even number of domain values along each edge this alternating bit assignment is always possible. Now observe, for a given edge, we can re-assign the bit-complement volumes to the domain points along the edge. The volumes associated with the nodes covering the edge nodes remain the same. I general, we can re-assign the bit-complement volumes to parallel edges independently of each other. In two dimensions this results in 2^2+2^2 additional symmetries. In d dimensions this results in $d \cdot 2^{2^{(d-1)}}$ additional symmetries. Composing them with the symmetries of the hypercube yields a lower bound of $2^{d+2^{(d-1)}}(d)(d!)$.

E.8 Proof of Theorem 16

Proof. We start by showing that each solution in $\mathcal{A}$ corresponds to a valid a valid database. Let G be the underlying DAG over domain $\mathcal{D}$. For correctness we require that for all nodes v in G.

$$x_v = \sum_{\substack{w \text{ sink of } G, \\ w.range \subseteq v.range}} x_w. \tag{5}$$

That is, for every node v in G, v's volume must be sum of the volumes assigned to the leaf-nodes that correspond to points in $v.range$. By Property (1), any non-sink node v of has a subset of children C such that $\{c.range : c \in C\}$ partition $v.range$. By Equation 3 there exists a constraint of the form $x_v = \sum_{c \in C} x_c$. Each $c \in C$, must itself also have a subset of children C' whose canonical rages partition $c.range$. By recursively substituting the variables corresponding to the children that partition the canonical range of each variable, until we reach the sinks (which are 1-1 with the points in the domain by Property (2)), we end up with the Equation 5. Each of the substituted equations are constraints in the ILP that are satisfied, so Equation 5 must also be satisfied. We conclude that any solution in $\mathcal{A}$ must be correspond to a real database.

Let $S_{\mathcal{L}}$ denote the reconstruction space and let $S_{\mathcal{A}}$ be the set of databases that correspond to solutions in $\mathcal{A}$. We will show that $S_{\mathcal{A}} = S_{\mathcal{L}}$. It is straightforward to see that $S_{\mathcal{L}} \subseteq S_{\mathcal{A}}$. In particular, since Equations 3 and 4 characterize the DAG G, then any database $D \in S_{\mathcal{L}}$ must satisfy the ILP.

Next we show that $S_{\mathcal{A}} \subseteq S_{\mathcal{L}}$. Since the databases in $S_{\mathcal{L}}$ are leakage-equivalent, then by Theorem 8 they result in the same volume map VM and frequency map FM, assuming each range query is issued exactly once. By Theorem 8 it is sufficient to show that the databases in $S_{\mathcal{A}}$ also result in VM and FM.

Let $\widehat{D} \in S_{\mathcal{A}}$ and let $\widehat{G}$ be its DAG of $\widehat{D}$ with volumes assigned to each node. Let $\widehat{\mathsf{VM}}$ and $\widehat{\mathsf{FM}}$ be the volume map and frequency map of $\widehat{G}$, respectively. From the leakage, we can build a map M mapping each observed tokenset $\mathbf{t}$ to its volume-frequency pair (vol, f). Equation 4 restricts each observed volume to be assigned to one node. The constraints impose a 1-to-1 correspondence between each volume with a given frequency f and each node in the DAG with frequency f. Since each observed tokenset has an associated volume-frequency pair, then the tokensets are 1-to-1 with the nodes in $\widehat{G}$; in particular each tokenset $\mathbf{t}$ such that $M[\mathbf{t}] = (vol, f)$ can be mapped to a node of $\widehat{G}$ with volume-frequency pair (vol, f).

Since each volume-frequency pair is 1-to-1 with the nodes in the $\widehat{G}$ of the same frequency then v must also have the same frequency and volume. Thus $\widehat{\mathsf{FM}}[\mathbf{t}] = \mathsf{FM}[\mathbf{t}]$. Also, we have that $\widehat{\mathsf{VM}}[\mathbf{t}] = (t, vol_{\mathbf{t}}) = \mathsf{VM}[\mathbf{t}]$. Since this holds true for all tokensets, then $\widehat{\mathsf{VM}} = \mathsf{VM}$ and $\widehat{\mathsf{FM}} = \mathsf{FM}$. This proves that Algorithm 8 achieves full database reconstruction.

The proof demonstrating that the input is available with probability $1 - \frac{1}{m^2}$ after the adversary observes $O(m^4 \log m)$ uniformly distributed queries follows from Lemma 5 in Appendix F. □

F ESTIMATING FREQUENCIES

In the SRC attack (Algorithm 8) we assume that each query is issued exactly once. This is a very strong assumption, and so in this section we show how an adversary can correctly estimate the frequencies with inverse polynomial probability in $O(m)$.

Lemma 5. *Let D be a d-dimensional database, over domain $\mathcal{D} = [m_1] \times \cdots \times [m_d]$, which is encrypted under the QDAG SRC scheme. Let $\widehat{F}$ be a dictionary mapping the observed search tokens to the number of times that search token was observed, G be the QDAG over $\mathcal{D}$. If the adversary observes N uniformly distributed queries, then the frequency of each search token* st *computed by Algorithm 10 (GetFrequencies) corresponds to the number of unique range queries*

Algorithm 10: GetFrequencies($\widehat{F}, G, \mathcal{D}$)

// $\widehat{F}$ is a dictionary mapping search tokens to the number of times each search token was observed, G is a DAG over domain $\mathcal{D}$.
Let N be the number of queries observed.
Let Q be the number of unique range queries over $\mathcal{D}$.
for st in $\hat{F}$ **do**
  $\widehat{F}[\text{st}] \leftarrow \widehat{F}[\text{st}]/(N/Q)$
return $\widehat{F}$

that are associated with st *happens with probability at least* $1 - 2|\widehat{F}|\exp(-2N/m^4)$*, where* $m = m_1 \times \cdots \times m_d$.

Proof. Suppose that the adversary has observed N queries being issued, and has constructed dictionary $\widehat{F}$. For each search token st observed define the i.i.d. random variable

$$X_i = \begin{cases} 1, & \text{if the } ith \text{ search token is st} \\ 0, & \text{otherwise.} \end{cases}$$

Let Z_{st} be the number of unique range queries that correspond to search token st and let Q is the number of unique range queries over $\mathcal{D}$. Observe that we have

$$\mathbb{E}[X_i] = \frac{Z_{\text{st}}}{Q}.$$

Define variable $A_{\text{st}} = \sum_i^N X_i$. We thus have that GetFrequencies succeeds when for all st we have

$$\frac{A_{\text{st}}}{N} \in \left[\frac{Z_{\text{st}}}{Q} \pm \varepsilon\right]$$

for $\varepsilon = O(1/Q) = O(m^{-2})$. Applying a Chernoff bound argument we see that

$$\Pr\left[\frac{A_{\text{st}}}{N} \geq N\left(\frac{Z_{\text{st}}}{Q} - \varepsilon\right)\right] \leq \exp(-2N\varepsilon^2).$$

A similar argument holds for the upper bound. Taking a union bound over the $|\widehat{F}|$ times we must approximate frequencies gives us a total success probability of $1 - 2|\widehat{F}|\exp(-2N/m^4)$. □