THE UNIVERSITY OF CHICAGO

A COMBINATORIAL APPROACH TO LEAKAGE ABUSE ATTACKS AND THEIR MITIGATION

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF DEPARTMENT OF COMPUTER SCIENCE

BY
FRANCESCA FALZON

CHICAGO, ILLINOIS
GRADUATION DATE

Dedication

Epigraph Text

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

ABSTRACT

With the rise of remote cloud services and the consequent rise in data breaches, there is an increased need for the secure outsourcing of data. The problem of enabling query processing over encrypted data without decryption is a challenging one, and approaches ranging from software to hardware solutions have been proposed. In this work, we take a closer look at a class of solutions that are efficient and deployable in the near-term future and that employ the use of light weight symmetric key primitives. In exchange for this added efficiency, these schemes leak certain information about the underlying data and queries. We explore the limitations of what a passive server-side adversary can learn from this information leakage and present practical constructions that minimize leakage while supporting complex queries.

In particular, we investigate the security of encrypted databases that support two types of expressive queries: range queries on multi-attribute data and shortest path queries on graph-structured data. The former is a common database query on relational data that requests all records whose attribute values lie within given query intervals. The latter is a fundamental graph query that returns the shortest path in a graph between two nodes, and has applications to the analysis of routing networks, metabolic networks, and social network. Multi-attribute data and graph-structured data are ubiquitous in our increasingly data-driven world, and understanding the security of how to encrypt them while also supporting expressive queries will help move us towards practical deployment.

In the first part of this work, we present the first attacks on schemes that support range queries over multi-attribute (or multi-dimensional) data. We describe the information theoretic limitation of reconstruction attacks in two settings and show that, in both cases, there can be an exponential number of distinct databases that produce equivalent leakage. We present a full database reconstruction attack that reconstructs the database when all queries are observed. We then relax these assumptions, and present an order reconstruction attack and an approximate database reconstruction attack that require only a strict subset

of the possible range queries to succeed.

In the second part of this work, we shift our focus to schemes that support shortest path queries on graph-structured data. We initiate our study by describing an attack that recovers the plaintext queries issued by the client on a graph encryption scheme, which we call the GKT scheme (Ghosh et al. AsiaCCS 2021). We then present a modified version of the GKT scheme with reduced leakage in exchange for a 2x increase in bandwidth overhead and a logarithmic increase in storage overhead. Our scheme uses data-structure techniques to decompose the graph into edge-disjoint paths. These paths are then stored in a multimap and encrypted using a standard encrypted multimap scheme. We support our scheme with a detailed cryptanalysis and explain why this new approach mitigates our previous attack.

CHAPTER 1

INTRODUCTION

> **Thesis statement:** Systems that enable the efficient processing of complex queries over encrypted data often leak information about the underlying data and their queries. We take a combinatorial approach to cryptanalyzing this leakage and mitigating these attacks.

Data breaches have been occurring with alarming frequency in the last few years, with billions of accounts being compromised in the Yahoo, Alibaba, and LinkedIn data breaches alone [15]. It comes, perhaps, as little surprise that these data can be highly sensitive. Now a days, nearly every piece of information – at all organizational levels – is stored on a personal computer or back-end server. As we acquire and store increasingly larger amounts of data, we find ourselves outsourcing our data to cloud service providers such as Google Cloud or Apple iCloud. This data ranges from electronic health records to personal banking details to confidential research data to classified government files. When such information is compromised, the effects, such as identity theft and financial fraud, can be deterimental.

Encryption can mitigate the risk of a data breach when data is outsourced to a third-party cloud service. One simple solution would be to encrypt the data as a single block and outsource it to the cloud. Querying the data would require one of two approaches. (1) The server has access to the decryption key: When a query is issued, the server decrypts the data, searches for and returns the response to the client, and re-encrypts the data. (2) The server does not have access to the decryption key: When a query is issued, the server returns the encrypted data to the client and the client decrypts the data and searches for the response. In the former, the server would still have access to the plaintext data and so would not prevent a passive server-side adversary, such as a curious cloud service employees, from reading the data. In the latter, the bandwidth requirements for even moderately-sized databases would

quickly become exorbitant and impractical. This is especially true given that bandwidth from cloud providers is much more expensive than storage. This strawman solution thus highlights the need for a better approach that is both more secure and more bandwidth efficient.

Encrypted databases (EDBs) provide a practical solution for strongly mitigating network and server-side attacks. EDBs have been extensively studied, and many solutions have been suggested to enable server query processing over encrypted data on behalf of clients. One potential solution is to use heavy cryptographic solutions such has ***fully-homomorphic encryption*** [50] or ***oblivious RAM (ORAM)*** [54]. While these provide strong security guarantees, they are still not as efficient and do not scale as well as we would like despite many recent advances. Another solution is to use ***trusted hardware*** solutions, such as Intel SGX [8, 63, 92]. However, while readily deployable with today's technologies, trusted hardware has been shown be vulnerable to powerful timing attacks [122], attacks leveraging rogue data cache loads that can extract secret keys [18], and code-reuse attacks that do not even require kernel privilege [10]. Hardening applications against such attacks often requires re-writing the application code [16] or using ORAM [119]. More over, trusted hardware still requires trusting a third party i.e. the company that manufactures the hardware.

The third class of solutions, which we refer to as ***structured encryption (STE)*** [29], is what this thesis is concerned with. Such solutions employ a variety of cryptographic primitives (often light weight symmetric key primitives) to encrypt structured data. This structured data may take on various forms ranging from document-oriented data to tree-structured data to matrix-structured data. Importantly, STE schemes offer sub-linear search on the encrypted data and thus the encrypted data can be queried for by the client without the server needing to decrypt the records. In 2022 MongoDB was the first database platform to release an EDB service that supports expressive queries on randomized encrypted data. With the deployment of EDBs in the wild such as MongoDB's "Queryable Encryption" it is more important than ever to understand the security of such schemes.

The efficiency of STE schemes, however, comes at a cost: the schemes are inherently "leaky" and reveal certain information about the underlying data or the query issued whenever a query is processed. Such leakage can include seemingly benign information such as the ***access pattern*** (i.e. pattern of memory locations accessed when retrieving the response to a query), ***search pattern*** (i.e. equality between queries), and ***volume pattern*** (i.e. the number of encrypted records returned with each query). Starting with the seminal work of Islam, Kuzu, and Kantarcioglu [64] in which they describe the first attack against searchable encryption schemes using access pattern, there have been a number of attacks that leverage leakage to reconstruct the database or queries (e.g. [40, 43, 70, 75]). The state of the art is still far from being as secure as we would hope. The question of Whether existing practical STE schemes are secure remains an important open question in applied cryptography. More to the point, understanding the weaknesses of existing STE schemes and learning how these weaknesses can be exploited is a crucial first step towards building better, more secure schemes. In particular, we seek to better understand the more complex queries that these schemes must support in order to be useful in today's world of big data. We describe two of the most important queries in the paragraphs below.

Range queries on multi-attribute data. Range queries are a fundamental query type that requests all records such whose value of a particular attribute (or attributes) lies within a given closed interval. All prior work on range queries, both on the attack side [58–60, 70, 73] and constructive side, have focused on range queries over one attribute. This thesis initiates the study of structured encryption schemes that support private ***range queries*** over two-attribute (two-dimensional) data. A range query over two attributes has the form:

```
SELECT * FROM T
WHERE (years BETWEEN 2010 AND 2020)
AND (avg_temp BETWEEN 15 AND 18)
```

where T is a table and `years` and `avg_temp` are attributes. In order to move towards practical EDBs, it is necessary that we better understand the security that such schemes provide.

Shortest path queries on graph-structured data. Graphs are an important mathematical object that can be used to visualize data and compute statistics about the data. Graph-structured data have been used in numerous settings including PageRank, social networks, fraud detection to biological networks, and recommendation systems – to name a few. NoSQL (non-relational) databases store and support queries on non-tabular data such as graphs. Given the importance of graph-structured data, graph database systems such as Amazon Neptune [5], Facebook TAO [17], Neo4j [97], and GraphDB [99] have risen in popularity. Despite the major role that graph-structured data and plaintext graph database systems play in real-world data management, our understanding of graph encryption schemes (GESs) is limited. Few GES have been described, most can only support one query type, and the recent attacks on existing schemes call into question their security. The second half of my thesis concerns itself with schemes that support single-pair shortest path queries on graphs. Formally, such a query takes as input a graph $G = (V, E)$ and two vertices $(u, v) \in V \times V$ and returns the shortest path between u and v in G, if it exists.

This thesis aims at advancing our understanding of STE schemes so that we can build practical schemes that support complex queries without compromising security. In particular, we take a close look at schemes that support range queries over multi-attribute (multi-dimensional) datasets and shortest path queries over graphs. In Chapters 3 and 4 we present the first attacks against schemes that support range queries in more than one dimension. In Chapter 5 we present the first attack against a scheme that supports shortest path queries on graphs. We conclude this thesis by then describing a novel scheme for supporting shortest path queries in Chapter 5; this scheme provably mitigates our attack while still boasting asymptotic efficiency. All our work includes theoretical analysis and experiments on real-world datasets to demonstrate the practicality of our attacks/scheme.

1.1 Thesis Overview

We now describe the organization of this thesis. The general trajectory of my work has been to first understand the weaknesses of existing schemes, develop attacks against these schemes, and then use the lessons learned to propose novel schemes that provably mitigate the attacks. My thesis specifically looks at two expressive and complex query types important in modern databases: range queries over multi-attribute data and shortest path queries on graph-structured data. The technical contributions of this thesis are divided into four chapters as follows.

- Chapter 3 presents the paper "Full Database Reconstruction Attack in Two Dimensions," which first appeared in CCS 2020 [43]. This work exploits access pattern leakage along with known query distribution or search pattern leakage to achieve full database reconstruction (which asks the adversary to recover the exact domain values of each encrypted record). This is the first attack to go beyond one dimension, and explore the the security of schemes supporting range queries in higher dimensions. One-dimensional databases can be reconstructed up to reflection. In contrast, we show that reconstruction in higher dimensions is significantly more complex and that the size of the set of two-dimensional databases that produce the same leakage profile can be exponential in the number of records. We present a complete polynomial-time attack that returns a polynomial-size encoding of all databases consistent with the given leakage profile.

- One natural question that arose from the work in Chapter 3 was whether one could launch a database reconstruction attack using fewer queries. This question gave way to Chapter 4, which covers our follow-up paper "Reconstructing with Less: Leakage Abuse Attacks in Two Dimensions," which appeared in CCS 2021 [87]. We considered two new database reconstruction goals: order reconstruction (which asks the adversary to recover the relative orders of the encrypted records) and approximate database

reconstruction (which asks the adversary to recover the domain values of each record to within some small error of the true value). Our order reconstruction attack requires only access pattern, and we experimentally demonstrate that reconstructing the order in higher dimensions can enable the attacker to infer the geometry of the underlying data – especially in more dense databases. Our approximate database reconstruction attack is distribution-agnostic and succeeds given any subset of the possible search pattern and the order of the records in the database. We additionally show how auxiliary statistical information of a related dataset can further improve reconstruction.

- In Chapter 5, we present "An Efficient Query Recovery Attack Against a Graph Encryption Scheme," which appeared in ESORICS 2022 [45]. Ghosh, Kamara and Tamassia presented a graph encryption scheme supporting shortest path queries [52]. We show how to perform a query recovery attack against this scheme when the adversary is given the original graph together with the leakage of certain subsets of queries. Specifically, the goal of the adversary is to recover the plaintext values of the queries issued – up to any information theoretic limitations – given the setup and query leakage. Our attack falls within the security model used by Ghosh et al., and is the first targeting schemes supporting shortest path queries. For a graph on n vertices, our attack runs in time $O(n^3)$ and matches the time complexity of the scheme's setup. We evaluate the attack's performance using the real world datasets used in the original paper.

- In light of the attack in Chapter 5, we set out to design a new secure graph encryption scheme that supports single-pair shortest path queries. In Chapter 6, titled "A Graph Encryption Scheme for Single-Pair Shortest Path Queries with Less Leakage", we present a scheme that employs classic data-structure techniques to reduce query leakage and mitigate our described attack. Our scheme only incurs an additional logarithmic factor in storage overhead over the GKT scheme, for a total of $O(n^2 \log n)$ storage where n is the number of vertices in the graph. Our scheme leverages classic graph algorithm

techniques to compute spanning subtrees whose paths correspond to shortest paths in the original graph and then decompose those trees into edge disjoint paths; a logarithmic number of copies of each edge disjoint path is stored in a multimap and then encrypted. For any query, the response size is guaranteed to be at most x2 the size of the queried shortest path, however our experimental evaluation on real world datasets demonstrates that, in practice, the size of the response is nearly optimal.

CHAPTER 2

COMMON PRELIMINARIES

Notation. For some integer n, let $[n] = \{1, 2, \ldots, n\}$. or an integer N let $[N] = \{1, 2, \ldots, N\}$. We denote concatenation of two strings a and b as $a||b$.

Dictionaries and Multimaps. A ***dictionary*** D is a map from a label space $\mathbb{L}$ to a value space $\mathbb{V}$. A ***multimap*** is a generalization of a dictionary in which each label may be associated with multiple values. Formally, a multimap M is a map from a label space $\mathbb{L}$ to the powerset of a value space $2^{\mathbb{V}}$. If $\mathsf{lab} \mapsto \mathsf{val}$ then we write $\mathsf{val} \leftarrow \mathsf{D}[\mathsf{lab}]$. We denote the assignment of val to lab as $\mathsf{D}[\mathsf{lab}] \leftarrow \mathsf{val}$ (and correspondingly for multimaps).

2.1 Graphs

Throughout this thesis, we employ the use of graphs both as a tool to carry out our attacks and as a way to structure data that we wish to encrypt. A ***graph*** is a pair $G = (V, E)$ consisting of a vertex set V of size n and an edge set E of size m. A graph is ***simple*** if the pairs of vertices in E are unordered ans a graph is ***directed*** if the edges specify a direction from one vertex to another. Two vertices $u, v \in V$ are ***connected*** if there exists a path from u to v in G.

A ***tree*** is a connected, acyclic graph. A ***rooted tree*** $T = (V, E, r)$ is a tree in which one vertex r has been designated the root. For some rooted tree $T = (V, E, r)$ and vertex $v \in V$ we denote by $T[v]$ the subtree of T induced by v and all its descendants.

A ***digraph*** is a tuple $G = (V, E)$ such that V is the vertex set and E is the *directed* edge set. For any two vertices or nodes $u, v \in V$ we denote a directed edge from u to v as the pair (u, v). A ***source vertex*** is a vertex with only outgoing edges and a ***sink vertex*** is a vertex with only incoming edges.

Given a graph $G = (V, E)$ and some vertex $v \in V$, we define a ***single-destination***

shortest path (SDSP) tree for v to be a directed spanning tree T such that T is a subgraph of G, v is the only sink in T, and each path from $u \in V \setminus \{v\}$ to v in T is a shortest path from u to v in G. An example of an SDSP tree can be found in Figure 5.1c.

We also define two binary operations on graphs. Given two graphs $G = (V, E)$ and $H = (V', E')$, the union of G and H is defined as $G \cup H = (V \cup V', E \cup E')$. Given a graph $G = (V, E)$ and a subgraph $H = (V', E')$ such that $V' \subseteq V, E' \subseteq E$, the graph subtraction of H from G is defined as $G \setminus H = (V \setminus V', E \setminus E')$.

2.1.1 Graph Isomorphisms

Our attack in Chapter 5 makes heavy use of graph isomorphisms and automorphisms.

Definition 1. *An **isomorphism of graphs** $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a bijection between vertex sets $\varphi : V_1 \to V_2$ such that for all $u, v \in V_1, (u, v) \in E_1$ if and only if $(\varphi(u), \varphi(v)) \in E_2$. If such an isomorphism exists, we write $G_1 \cong G_2$.*

Definition 2. *An **isomorphism of rooted trees** $T_1 = (V_1, E_1, r_1)$ and $T_2 = (V_2, E_2, r_2)$ is an isomorphism φ from T_1 to T_2 (as graphs) such that $\varphi(r_1) = r_2$.*

2.1.2 Canonical Names

A ***canonical name*** is an encoding $\mathsf{Name}(\cdot)$ mapping graphs to bit-strings such that, for any two graphs H and G, $\mathsf{Name}(G) = \mathsf{Name}(H)$ if and only if $G \cong H$. For rooted trees Aho, Hopcraft, and Ullman (AHU) [3] describe an algorithm for computing a specific canonical name in $O(n)$ time. We refer to this as ***the*** canonical name and describe it next.

The AHU Algorithm. We use a modified AHU algorithm, denoted as ComputeNames, to compute the canonical names of rooted trees (and their subtrees) and determine if they are isomorphic. ComputeNames takes as input a rooted tree $T = (V, E, r)$, a vertex $v \in V$, and an empty dictionary Names. It outputs the canonical name of the subtree $T[v]$ (which we

also refer to as the canonical name of v) and a dictionary Names that maps each descendent u of v to the canonical name of $T[u]$. The algorithm proceeds from the leaves to the root. It assigns the name '10' to all leaves of the tree. It then recursively visits each descendent u of v and assigns u a name by sorting the names of its children in increasing lexicographic order, concatenating them into an intermediate name *children_names* and assigning the name '1||*children_names*||0' to u (see Figure 5.1b for an example). The canonical name of T, Name(T), is the name assigned to the root r by this algorithm.

Our implementation of the algorithm returns a dictionary Names mapping each descendent u of v to the short canonical name of the subtree $T[u]$. Maintaining the dictionary Names enables us to compute the canonical names of the subtrees rooted at each $v \in V$ in a single traversal of T. ComputeNames takes time and space $O(n^2)$ where $|V| = n$. Note that the original AHU algorithm can be modified to run in $O(n)$ time, by only considering one level at a time and reassigning integers to the vertices at that level [3]. In contrast, we need to assign names to each vertex in the tree in order to later compute the path names.

The pseudocode of ComputeNames can be found in Algorithm 1.

Algorithm 1 [3] ComputeNames

Input: Rooted tree $T = (V, E, r)$, vertex $v \in V$, and dictionary Names.
Output: Dictionary Names.

```
1: if v is a leaf then
2:     Names[v] = "10"
3:     return Names
4: else
5:     Initialize empty list temp = [ ]
6:     for child u of v do
7:         Names ← ComputeNames(T, v, Names)
8:         temp.append(Names[u])
9:     Sort temp in ascending order
10:    Concatenate names in temp as children_name
11:    Names[v] = "1"||children_name||"0"
12:    return Names
```

2.1.3 Heavy-Light Decomposition

Heavy-light decomposition (HLD) was introduced by Sleator and Tarjan in order to develop fast algorithms for a number of tree problems, including computing nearest common ancestors and various network flow problems [116]. Before discussing the method, we introduce a few definitions.

Definition 3. *Given a rooted tree T, the* ***size of a node*** *v* ***in*** *T, $\mathsf{size}_T(v)$, is the number of nodes in the subtree rooted at node v (the size includes the node v itself).*

Definition 4. *Let T be a rooted tree. An edge between a node v in T and its parent is defined as* ***heavy*** *if and only if $\mathsf{size}_T(v) \geq \frac{1}{2}\mathsf{size}_T(\mathsf{parent}(v))$. All other edges in the tree are* ***light****.*

By definition, any node in a tree has at most one child linked to by a heavy edge.

Definition 5. *The heavy edges decompose the tree nodes of a tree T into vertex disjoint paths which are called* ***heavy chains****. These paths are connected to each other via light edges.*

Theorem 2.1.1. *([116]) Let T be a rooted tree with n nodes. From any node in T, the number of light edges needed to reach the root of the tree is at most $\log n$. Thus, the number of heavy chains along the path from any node to the root is also $O(\log n)$.*

Complexity. This algorithm only marks each edge while exploring the input tree T using DFS. The running time is thus the same time and space as running DFS on a tree i.e. $O(n)$ where n is the number of nodes in T.

Algorithm 2 [116] ComputeHLD

Input: Rooted tree $T = (V, E, r)$, and vertex $v \in V$.
Output: Size of subtree $T[v]$ and tree T with edges labeled as either "heavy" or "light".

if v is a leaf **then**
    // The subtree rooted at v is of size 1.
    return $1, T$
else
    $v\_size = 1$
    $temp = \{\}$
    // Compute size of subtree rooted at v.
    for child w of v **do**
        $w\_size, T \leftarrow$ ComputeHLD(T, w)
        $temp[w] \leftarrow w\_size$
        $v\_size \leftarrow v\_size + w\_size$
    for (w, w_size) in $temp$ **do**
        // Determine if (w, v) is heavy or light.
        if $w\_size < v\_size/2$ **then**
            Label (w, v) in T as "light"
        else
            Label (w, v) in T as "heavy"
    return v_size, T

2.2 Two-Dimensional Databases

In Chapters 3 and 4, we fix positive integers N_0, N_1 and let the domain $\mathcal{D}$ be defined as $\mathcal{D} = [N_0] \times [N_1]$. When $N_0 = N_1$ we say that $\mathcal{D}$ is ***square***. We call ***main diagonal of*** $\mathcal{D}$ the set of points that lie on line segment from $(0, 0)$ to $(N_0 + 1, N_1 + 1)$. For a point $w \in \mathcal{D}$, we write w_0 for its first coordinate (horizontal) and w_1 for its second coordinate (vertical), so $w = (w_0, w_1)$. We also recall the geometric concept of dominance between points: point $w \in \mathcal{D}$ ***dominates*** point $x \in \mathcal{D}$ if $x_0 \leq w_0$ and $x_1 \leq w_1$. We denote this as $x \preceq w$. Similarly, point $w \in \mathcal{D}$ ***anti-dominates*** point $x \in \mathcal{D}$ if $w_0 \leq x_0$ and $x_1 \leq w_1$, and we denote this as $x \preceq_a w$. The dominance or anti-dominance is said to be ***strict*** if the above inequalities are strict. We say that w ***minimally (anti-) dominates*** x if there is no point $v \neq w, x$ such that w (anti-) dominates v and v (anti-) dominates x.

We define a ***2-dimensional database*** D ***over domain*** $\mathcal{D}$ as an element of $\mathcal{D}^R$ for some integer $R \geq 1$, i.e., an R-tuple of points in $\mathcal{D}$. We refer to the entries of D as ***records***. We

call the ***identifier*** (or ***ID***) of a record its index in the tuple (an integer $j \in [R]$). Also, the domain value associated with ID j is denoted $D[j]$. Note that the same value in $\mathcal{D}$ may be associated with multiple database records. In the rest of this thesis, for simplicity, whenever it is clear from the context, we may refer to records of a database as points.

Range queries and responses. A range query returns the identifiers of the records whose points are in a given range. Formally, a ***range query*** is a pair $q = (c, d) \in \mathcal{D}^2$ such that $c \preceq d$. We define the ***response*** or ***access pattern*** of $q = (c, d)$ to be the set of identifiers of records in D whose points lie in the rectangle "between" c and d. Formally,

$$\mathsf{Resp}(D, q) = \{j \in [R] \ : \ c \preceq D[j] \preceq d\}. \tag{2.1}$$

We define the ***response multiset of a database*** D, denoted $\mathsf{RM}(D)$, as the ***multiset*** of all access patterns of D:

$$\mathsf{RM}(D) = \{\{\mathsf{Resp}(D, q) : q = (c, d) \in \mathcal{D}^2, c \preceq d\}\}. \tag{2.2}$$

The double bracket notation emphasises the multiset as distinct queries q and q' may produce the same response, $\mathsf{Resp}(D, q) = \mathsf{Resp}(D, q')$. For two multisets A and B, we say that A is a ***submultiset*** of B if A is contained in B. We define the ***response set*** of D, denoted $\mathsf{RS}(D)$, to be the set associated with $\mathsf{RM}(D)$ where each response appears exactly once.

Computing the response multiset. Our algorithms assume the response multiset $\mathsf{RM}(D)$ as input. This allows us to isolate the combinatorial and geometric structure of the problem. We now show how an adversary can calculate $\mathsf{RM}(D)$ with access pattern leakage plus (i) known query distribution, (ii) search pattern leakage and known query distribution, or (iii) search pattern leakage and known database size. These are all standard in previous work.

In case (i), the adversary computes each unique response s of $\mathsf{RM}(D)$ and its multiplicity, which is given by the probability of s being returned by a query. For example, given a uniform

distribution of queries, the multiplicities can be computed with high probability using a standard Chernoff bound argument after roughly $O(N^4)$ queries. Similar techniques can be used for other distributions.

In cases (ii) and (iii), the adversary can derive $\mathsf{RM}D)$ after observing a response to every query at least once. To know when this has occurred, the adversary waits for a sufficient number of queries that depends on the query distribution (case (ii)) or until all distinct queries have been seen, their count based on the size of the database (case (iii)). Notably, in cases (ii) and (iii), if the queries are uniform, $O(N^2 \log N)$ queries are sufficient to compute $\mathsf{RM}(D)$ with high probability, by the coupon collector argument.

2.2.1 Leakage Equivalent databases

We now introduce the notion of equivalent databases.

Definition 6. *Databases D and D' are* ***equivalent with respect to the response multiset*** *if $\mathsf{RM}(D) = \mathsf{RM}(D')$ and* ***equivalent with respect to the response set*** *if $\mathsf{RS}(D) = \mathsf{RS}(D')$.*

Both equivalence definitions imply that the databases the same number of records. When it is clear from context which type of equivalence we are referring to, we may simply refer to "equivalent databases".

2.2.2 Full database reconstruction

We denote the set of databases equivalent to D with respect to the response multiset as

$$\mathsf{E}(D) = \{D' \in \mathcal{D}^R : \mathsf{RM}(D) = \mathsf{RM}(D')\}. \tag{2.3}$$

We define full database reconstruction as follows:

Definition 7. ***Full Database Reconstruction (FDR).*** *Given the multiset $\mathsf{RM}(D)$ for some database D, compute $\mathsf{E}(D)$.*

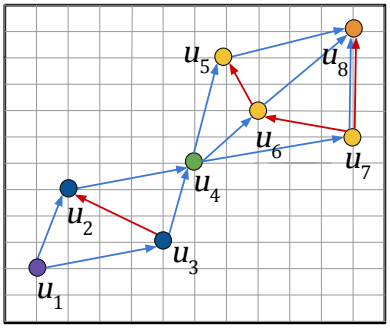


Figure 2.1: The dominance graph (blue) and anti-dominance graph (red) for a database with 8 records and components $\{u_1\}$, $\{u_2, u_3\}$, $\{u_4\}$, $\{u_5, u_6, u_7\}$, and $\{u_8\}$.

Computing $\mathsf{E}(D)$ is the best an adversary can do without prior information on D or the queries. We revisit the setting later and show that, with training data, an adversary can often recover D after computing $\mathsf{E}(D)$ by looking for the most typical member.

2.2.3 Order Reconstruction

In order to define order reconstruction, we must first introduce the concept of (anti-)dominance graphs. The definitions below are illustrated in Figure 2.1.

Definition 8. *The **dominance graph**, $G = (V, E)$, of a set of points S, is the digraph where $V = S$ and $(a, b) \in E$ if b minimally dominates a and $a, b \in V$.*

Definition 9. *The **anti-dominance graph**, $G' = (V', E')$, of a set of points S, is the digraph where $V' = S$ and $(a, b) \in E'$ if b minimally anti-dominates a and $a, b \in V'$.*

Let D be a database and let G and G' be the dominance and anti-dominance graphs of D, respectively. We define the set of all possible point orderings of databases equivalent to D with respect to response set, $\mathsf{RS}(D)$ as $\mathsf{E_o}(D)$. Formally,

$$\mathsf{E_o}(D) = \{D' \in \mathcal{D}^R : \mathsf{RS}(D) = \mathsf{RS}(D')\}. \tag{2.4}$$

We define ***order reconstruction (OR)*** as follows:

Definition 10. ***Order Reconstruction (OR).*** *Given a set* $\mathsf{RS}(D)$ *of some database* D*, compute all pairs of dominance and anti-dominance graphs* (G, G') *such that any database* D' *with record relationships defined by* (G, G') *is equivalent to* D *with respect to the response set, i.e.* $\mathsf{RS}(D) = \mathsf{RS}(D')$*.*

Computing (G, G') is the information theoretic best that an adversary can do without additional information.

2.2.4 Approximate Database Reconstruction

The problem of ***approximate database reconstruction (ADR)*** is defined as follows. Given the order of points in D and a multiset of search token and response pairs (where each pair corresponds to one of the observed queries), (i) estimate the number of unique queries that each record appears in and then (ii) use this information to construct a system of non-linear equations that can be solved to give approximate values of the records.

2.3 Prior and Related Work

2.3.1 Searchable Symmetric Encryption and Structured Encryption

Key word search over encrypted outsourced documents is a fundamental problem in data management. The goal is typically to build and encrypt an inverted-index that maps each key word to the set of matching document identifiers such that that the server can process key word queries over the encrypted data. The search should be sub-linear in the size of the data. ***Searchable symmetric encryption (SSE)*** presents a solution to this problem and there exists a long line of work that formalizes the notion of SSE and presents increasingly secure and efficient solutions such as [13, 14, 23, 24, 29, 33, 34, 51, 68, 69, 96, 117] and see also the survey by Fuller et al. [48]. All existing SSE schemes make an important trade-off between security and efficiency: when a query is processed, the client learns the results of the query

and the server learns some small well-defined information about the underlying data and the query. This "leakage" can include information such as access pattern or volume patters; we discuss the implications of this leakage in following subsections.

A more recent line of work has proposed and analyzed the security of volume-hiding schemes [7,66,102,126,127] that reduces this leakage by hiding the number of records returned. These schemes generally pad the responses in order to hide the volume at the expense of a small constant factor increase in the bandwidth and storage overhead.

Structured encryption (STE) was introduced by Chase and Kamara in [29] as a way to generalize SSE. STE enables the encryption of structured data (e.g. a social network or a dictionary) in such a way that search on this data is both private and efficient. search for a variety of query types beyond key-word search, including graph queries (e.g. [29, 52]) and subsets of SQL (e.g. [42, 65]). These approaches have been used both in data management research (e.g., [101, 103, 106]) and industry (e.g., [30, 91]).

2.3.2 Range Supporting Schemes

Order revealing encryption (ORE) [2, 11, 12] has been used to support range queries, but even the most-secure "ideal" ORE constructions leak the order of the records before even being queried, and ORE schemes have been shown vulnerable to devastating ***leakage abuse attacks*** (e.g., [9, 40, 56, 95]) that recover data in certain circumstances, a fact that underscores the need to better understand the security of these schemes. In contrast, the encrypted records stored in range schemes built from ***Searchable symmetric encryption (SSE)*** do not inherently leak such order information; in general, these schemes build an index that maps range queries to the records (or index oof the records) that match the range, and the index is then encrypted using a standard SSE scheme. To respond to a range query, the client may query for a specific range or a collection of subranges that correspond to the range query of interest. Most efficient range schemes built from SSE only support range queries

on only single-attribute (1D) data. Demertzis et al. [37, 38] present one-dimensional range schemes with storage and security trade-offs, and Faber et al. [42] builds on the SSE scheme in [24] to support 1D range, substring, wild-card, and phrase queries. Falzon et al. [44] are the first to construct schemes that also support multi-dimensional queries over symmetrically encrypted data.

2.3.3 Leakage Abuse Attacks against Range Supporting Schemes

The following attacks described in this subsection were carried out by passive adversaries against a one-dimensional database with the domain $[1, N]$ for some integer N. Kellaris et al. [70] were the first to give such an attack from the leakage of range queries. They showed that given the access pattern of the queries, one could determine the exact record values up to reflection after observing $O(N^4 \log N)$ uniformly random queries. We refer to the goal of recovering the exact values of all the records as ***full database reconstruction (FDR)***. Kellaris et al. further showed that FDR can be achieved with only $O(N^2 \log N)$ queries if the database is ***dense***. Informally, a dense database is one in which each domain value is associated with at least one record. Since this seminal work, a number of other papers have explored the problem in one-dimension (e.g. [59, 72, 73, 75, 88]). In [75], Lacharitè et al. improve on the dense database attack and present an algorithm that succeeds in FDR of dense databases with only $O(N \log N)$ uniformly random queries.

Order reconstruction was introduced in [70] as the first step of their FDR attack. Grubbs et al. [59] generalize the attack to achieve sacrificial ϵ-approximate order reconstruction (ϵ-AOR) where $\epsilon > 0$ is the chosen precision factor. The goal of sacrificial ϵ-AOR is to recover the order of all records, except for records that are either within ϵN of each other or within ϵN of the domain endpoints. Their attack achieves sacrificial ϵ-AOR with probability $1 - \delta$ for some $\delta > 0$ given the access pattern of $O(\epsilon^{-1} \log \epsilon^{-1} + \epsilon^{-1} \log \delta^{-1})$ uniformly random queries. Furthermore, setting $\epsilon = 1/N$ achieves FDR.

Approximate database reconstruction from access pattern of range queries in one-dimension was addressed in [59, 73, 75]. Lacharité et al. [75] introduce ϵ-approximate database reconstruction (ϵ-ADR) as the goal of reconstructing each record value up to ϵN error for some $\epsilon > 0$. They give an attack that achieves ϵ-ADR with $O(N \log \epsilon^{-1})$ uniform queries. In [59], the authors describe an attack that achieves sacrificial ϵ-ADR; the goal is to recover all values up to an error of ϵN, while "sacrificing" recovery of points within ϵN of the domain endpoints. Concepts from statistical learning theory are applied to achieve a scale-free attack that succeeds with only $O(\epsilon^{-2} \log \epsilon^{-1})$ queries.

Kornaropoulos et al. [73] reconstruct a one-dimensional database without knowledge of the underlying query distribution and without observing all possible queries by employing statistical estimators to approximate the support size of the conditional distribution of search tokens given a particular response. Their agnostic reconstruction attack achieves reconstruction with good accuracy in a variety of settings including and beyond the uniform query distribution.

There are a number of reconstruction attacks [58, 60, 70, 74] that only use ***volume pattern*** i.e., the number of records matching each query. The goal of these attacks is to reccover the number of records associated with each domain value. Kelaris et al. [70] initiate the study of attacks using only volume pattern; their attack builds a polynomial from the observed volumes and then factors it into two polynomials to recover the database volumes. Grubbs et al. [58] give an attack that builds a graph from the leakage and then uses clique-finding combined with heuristics to recover the database. The attack by Gui et al. [60] uses a greedy approach to recover the database volumes from the leakage of range queries with bounded width b. Most recently, Kornaropoulos et al. [74] give an attack that employs a combination of counting functions and constrained optimization solvers.

2.3.4 Graph Encryption Schemes

We now describe prior and related work concerning graph encryption schemes. Chase and Kamara [29] demonstrate the generality of STE by describing schemes that support adjacency, neighbor, and focused subgraph queries on graphs. Meng et al. [93] present three schemes of varying leakage levels and efficiencies. To reduce storage overhead, their schemes leverage ***sketch-based oracles*** that select seed vertices and store the exact shortest distance from all vertices to the seeds; these distances are then used to estimate shortest paths between any two vertices in the graph. Wang et al. [128] utilize additively homomorphic encryption and garbled circuits to encrypt graph data and support shortest distance queries. Ghosh et al. [52] describe a scheme based on the SP-matrix of the graph and whose setup time, query time, and storage are asymptotically optimal.

A number of STE schemes supporting other query types and graphs have been described such as GES supporting top-k-nearest neighbors [82], STE for conceptual graphs [105], STE for knowledge graphs [130], and dynamic STE for bipartite graphs [76,80]. Solutions for graph queries in other security models have also been proposed (e.g., information retrieval [129] and differential privacy [111]), but are outside the scope of this work.

2.3.5 Leakage Abuse Attacks against Graph Encryption Schemes

The leakage of graph encryption schemes was first analyzed by Goetschmann [53]. The author considers schemes that support approximate shortest path queries that use sketch-based distance oracles (e.g. [93]), presents two methods for estimating distances between nodes, and gives a query recovery attack that aims to recover the vertices in an encrypted query; the experimental evaluation demonstrates that with auxiliary knowledge on some queries, the adversary can distinguish among candidate vertices which vertex was queried. Our attack which we describe in Chapter 5 is also a query recovery attack, but uses knowledge of the plaintext graph rather than partial knowledge of some queries.

CHAPTER 3

FULL DATABASE RECONSTRUCTION FROM RANGE QUERIES IN TWO DIMENSIONS

This chapter first appeared as a conference publication by the same title with minor modifications in CCS 2020 [43]. It is joint work with Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia.

3.1 Introduction

Several recent works [58, 59, 70, 72, 73, 75, 88] have presented attacks that leverage a mild-looking form of leakage on range queries and k-NN queries. The attacks assume knowledge of the ***access pattern*** of the query, meaning the identifiers of the records in the response of the query. With enough queries and knowledge of their distribution, these attacks can efficiently fully recover the plaintext data, up to reflection (intuitively, the attack is not sure about the order of the data, since increasing and decreasing sequences of points are indistinguishable in their models). Attacks can also leverage another type of mild-looking leakage, ***search pattern leakage***. Using search pattern leakage, the attacker can determine whether two identical responses correspond to the same query. The combination of access and search pattern leakage can achieve equivalent results without knowledge of the query distributions.

3.1.1 Contributions

We perform the first exploration of reconstruction attacks on encrypted databases that support ***two-dimensional range queries***. Concretely, we consider settings that allow for conjunctive range queries on two columns (e.g. a query q selects all records with weight between w_0 and w_1 and height between h_0 and h_1). We consider this problem because plausible efficient constructions can support such queries while leaking strictly less information than systems

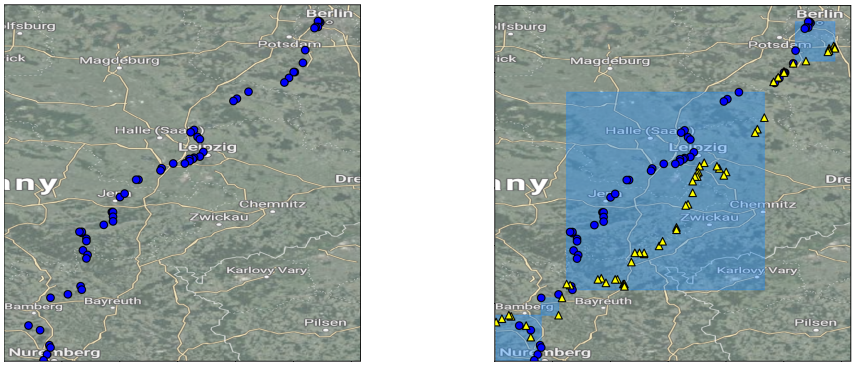


Figure 3.1: Reconstruction of Malte Spitz's [118] location data from 08/31/2009. Original locations are drawn as blue points and their reflections as yellow triangles. The points in each highlighted box and stand-alone pair can independently flip along the diagonal, producing 1024 equivalent databases.

that preform one-dimensional range queries. Prior work in one-dimension leaves the security of these systems open.

Our results assume a basic form of leakage where a persistent passive adversary learns which (encrypted) records are returned for each query. Our attack additionally requires either knowledge of the query distribution or search pattern leakage. Our leakage choice is conservative, and as we argue below, will apply to many potential approaches towards supporting two-dimensional range queries. It is also strong, in that (following prior work) we require all possible queries to be issued in order to get a clean theoretical understanding of possible attacks. In our setting we completely characterize what is information-theoretically possible to recover by the adversary, showing that in the two-dimensional setting it can be complicated even to describe. We then develop an efficient algorithm that succeeds in finding all databases consistent with the observed leakage.

There are many possible ways to support encrypted two-dimensional range queries. We selected a leakage profile that reflects a common-denominator leakage that appears difficult to efficiently avoid and is also technically interesting to attack. To begin understanding our setting, consider an index-based approach that independently supports range queries on individual columns, and simply translates a two-dimensional query into the intersection of

one-dimensional queries. Such a system, on the query q from the previous paragraph, will leak which columns have the requested weight and which columns have the requested height. Prior one-dimensional attacks (e.g. [59, 70]) can thus be applied to each dimension, recovering the full database.

A few approaches can leak strictly less than this one and render the one-dimensional attacks inapplicable. A conceptually simple system could, roughly speaking, precompute a joint index for all possible two-dimensional q. Once this index is encrypted (using an encrypted multimap [23, 29, 34], say), only the records matching both dimensions will be retrieved when processing a given q (in contrast to the naive solution, where records matching the weight range but not the height range would be accessed unnecessarily). Since the dimensions interfere with each other to produce leakage, prior attacks do not apply.

Another approach which could produce similar leakage is to use an oblivious primitive like ORAM to obtain the identifiers of records matching the query, and then access the actual records in a standard data store. Such an approach, which has been used for encrypted keyword searches (e.g., [49]), and dynamic constructions like [36, 51], is desirable when the actual records are large compared to the indexing information, as it would reduce the size of the ORAM. This approach hides the search pattern and the leakage on individual columns, but still reveals the access pattern of records matching the query. Thus, this method is vulnerable to our attack when the query distribution is known.

Maple is a system for multi-dimensional range search over encrypted cloud data [125]. Their approach focuses on not leaking single-dimension information. To achieve this, the system leaks, in addition to access and search pattern leakage, the path pattern of the search tree (which nodes were accessed on the multi-dimensional range search tree) and the values of each query (which ranges are being queried). More recently, Kamara et al. [65, 67] show how to perform conjunctive SQL queries with a reduced leakage profile, but only for a single value and not ranges.

In contrast to the one-dimensional case where complete recovery up to reflection is possible, we present an information-theoretic limit to the power of reconstruction attacks in two dimensions. Namely, we show that there exist exponentially-large families of different databases that have indistinguishable access and search pattern leakage. Also, for a database D, we fully characterize the set of databases with leakage identical to that of D in terms of combinatorial and geometric properties of D as well as number-theoretic considerations involving the domain of points of D, including the number of integral solutions to a certain Diophantine equation. We tame this complexity by providing a characterization and succinct encoding of this set of indistinguishable databases.

Based on this characterization, we exhibit a poly-time attack that recovers the set of indistinguishable databases returning a poly-space encoding of it. For a database with R records over a rectangular domain with $N_0 \times N_1$ points, where $N = N_0 \cdot N_1$, our attack takes time $O((N_0 + N_1)(RN^2 + R \log R))$. Our attack works for an arbitrary database, with no assumptions on the configuration of the points. In particular, we support both dense and sparse databases, allowing zero, one, or multiple records per domain point.

We have implemented our attack and evaluated it on several datasets of real-world health and location data. We illustrate in Figure 3.1 our reconstruction of a real-world location dataset by our attack, which recovers a family of equivalent databases obtained by independently flipping along the diagonal certain highlighted subsets of points. Finally, we developed another attack that, assuming some prior auxiliary knowledge, picks the "real" database out from amongst the indistinguishable set, and showed that it typically succeeds on real-world data.

We summarize our main contributions as follows:

1. We **characterize** the families of 2D databases with the same leakage profile and show they may contain an exponential number of databases (Section 3.3, Theorems 3.3.3 and 3.3.4).

2. We **develop** an efficient poly-time full database reconstruction attack that encodes the potentially exponential databases in poly-space (Section 3.4, Algorithm 7 and Theorem 3.4.5).

3. We **implement** the attack and **evaluate** it on real-world location and health data (Section 3.5).

4. Given access to training data, we show how to **reduce** the size of the solution set (Section 3.6).

This chapter is the product of merging two independent lines of work, [4] and [90].

3.2 Technical Tools and Overview

We introduce the main technical ingredients in our algorithm, and then provide an overview of how they are combined. At the end of this section we apply these tools to classify the structure of $\mathsf{E}(D)$ for any database D. We will then apply our classification in the analysis of our main algorithm.

Reflection. Symmetries will play a central role in understanding $\mathsf{E}(D)$, the most important of which for us is ***reflection***. We define the ***reflection*** of a point $w = (w_0, w_1)$ of domain $\mathcal{D} = [N_0] \times [N_1]$ to be the point $w' = (w'_0, w'_1)$ such that (see Figure 3.2)

$$w'_0 = w_1 \cdot \frac{N_0 + 1}{N_1 + 1}; \qquad w'_1 = w_0 \cdot \frac{N_1 + 1}{N_0 + 1}. \tag{3.1}$$

We refer to the reflection of a point using function $w' = \sigma(w)$. The reflection of w, $\sigma(w)$, can be obtained geometrically by considering the rectangle with horizontal and vertical sides that has one corner at point w and two other corners on the main diagonal of $\mathcal{D}$. The reflection, w' of w is the remaining corner of this rectangle.

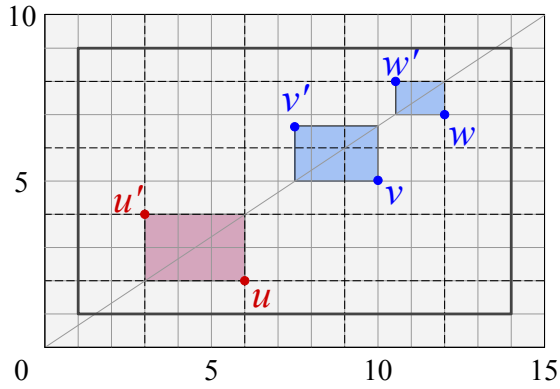


Figure 3.2: Points u, v and w of domain $\mathcal{D} = [14] \times [9]$ (thick black rectangle) and their reflections u', v' and w'. We have that $u' \in \mathcal{D}$ but $v' \notin \mathcal{D}$ and $w' \notin \mathcal{D}$. By Lemma 3.2.1, the points of $\mathcal{D}$ whose reflection is in $\mathcal{D}$ have coordinates of the type $(3i, 2j)$, i.e., are at the intersections of the dotted grid-lines.

Note that the reflection w' of w may or may not be in $\mathcal{D}$. The following lemma characterizes the points of $\mathcal{D}$ whose reflection is also in $\mathcal{D}$.

Lemma 3.2.1. *Let $w = (w_0, w_1)$ be a point of domain $\mathcal{D} = [N_0] \times [N_1]$ and let $\frac{\alpha_0}{\alpha_1}$ be the reduction of fraction $\frac{N_0+1}{N_1+1}$ to its lowest terms. We have that the reflection of w is in domain $\mathcal{D}$ if and only if w_0 is a multiple of α_0 and w_1 is a multiple of α_1.*

Our definition of reflection refers to the main diagonal of the domain. We can define a similar concept referring to the other diagonal, i.e., the line segment from $(N_0 + 1, 0)$ to $(0, N_1 + 1)$.

3.2.1 Query Densities

We will repeatedly use a strategy that generalizes the main observation of [70]. There, in trying to determine a point x, they observed that one can compute the proportion of $\mathsf{RM}(D)$ in which x appears. Then they could proceed algebraically to limit the number of possible values for x. In particular, in one dimension, this narrowed x down to two values. The final step of their algorithm reduced this to one possibility by fixing another point y and recording how often x and y appeared together, adding another constraint.

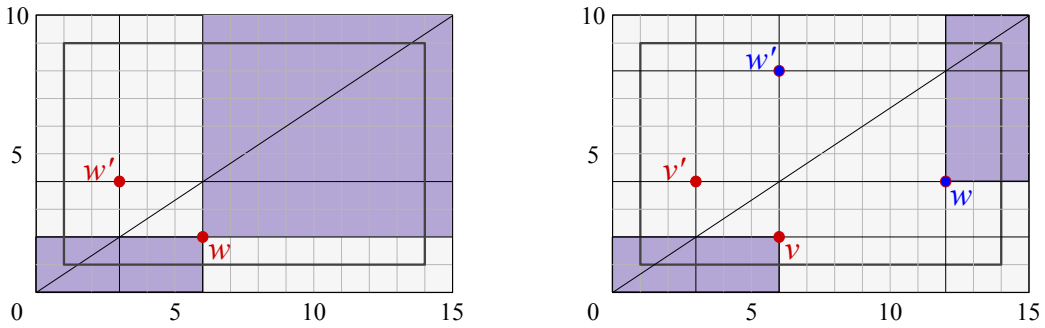


Figure 3.3: Illustration of Equations 3.2 and 3.3 for points of domain $\mathcal{D} = [14] \times [9]$. (a) The query density of point $w = (6, 2)$ is the product of the areas of the two rectangles, i.e., $\rho_w = 6 \cdot 2 \cdot (15 - 6) \cdot (10 - 2) = 12 \cdot 72 = 864$. Since the reflection $w' = (3, 4)$ of w is a point of $\mathcal{D}$, by Lemma 3.2.2, we have $\rho_{w'} = \rho_w = 864$. (b) The query density of pair $v = (6, 2)$ and $w = (12, 4)$ is the product of the areas of the two purple filled rectangles, i.e., $\rho_{v,w} = 12 \cdot 18 = 216$. Since the reflections $v' = (3, 4)$ of v and $w' = (6, 8)$ of w are points of $\mathcal{D}$, by Lemma 3.2.3, we have $\rho_{v,w} = \rho_{v',w} = \rho_{v,w'} = \rho_{v',w'} = 216$.

We now generalize the notion of query density to two dimensions. For a domain $\mathcal{D} = [N_0] \times [N_1]$, and $x \in \mathcal{D}$, define

$$\rho_x = \left|\{(c, d) \in \mathcal{D}^2 : c \preceq x \preceq d\}\right|$$

and for a pair of points $x, y \in \mathcal{D}$ define

$$\rho_{x,y} = \left|\{(c, d) \in \mathcal{D}^2 : c \preceq x, y \preceq d\}\right|.$$

These are the number of queries that contain x or x and y (respectively). Thus, the formula for the query density ρ_x of a point $x = (x_0, x_1) \in \mathcal{D} = [N_0] \times [N_1]$ is as follows (see Figures 3.3 and 3.5).

$$\boxed{\rho_x = x_0 x_1 (N_0 + 1 - x_0)(N_1 + 1 - x_1)} \tag{3.2}$$

Lemma 3.2.2. *Let w be a point of domain $\mathcal{D}$ and suppose the reflection w' of $\mathcal{D}$ is also in $\mathcal{D}$. We have that w and w' have the same query density, i.e., $\rho_{w'} = \rho_w$.*

Similarly, the formula for the query density $\rho_{v,w}$ of a pair of points, v and w, of $\mathcal{D}$ such that $v \preceq w$ is as follows.

$$\boxed{\rho_{v,w} = v_0 v_1 (N_0 + 1 - w_0)(N_1 + 1 - w_1)} \tag{3.3}$$

Again, we obtain the same query density by replacing one or both points of a pair with their reflections, as shown in Lemma 3.2.3. We note that this equation only holds when $v \preceq w$. If not, there are similar formulas depending on their (anti-) dominance relationship.

Lemma 3.2.3. *Let $v \preceq w$ be points of domain $\mathcal{D}$ and v' and w' be their reflections. We have*

$$\begin{aligned}
\rho_{v,w} &= \rho_{v',w} && \text{if } v' \in \mathcal{D} \text{ and } v' \preceq w \\
\rho_{v,w} &= \rho_{v,w'} && \text{if } w' \in \mathcal{D} \text{ and } v \preceq w' \\
\rho_{v,w} &= \rho_{v',w'} && \text{if } v' \in \mathcal{D},\ w' \in \mathcal{D} \text{ and } v' \preceq w'
\end{aligned}$$

Our attack exploits the fact that given response multiset $\mathsf{RM}(D)$, one can compute the query densities of all the points and pairs of points of the database without knowing their coordinates.

Two technical lemmas. The following lemmas will be used in both our classification of the structure of $\mathsf{E}(D)$ and in the analysis of our algorithm. They will be applied to infer where a point (or points) must be located based on query density constraints. The first bounds the number of points w that satisfy $\rho_w = \alpha$ below the trivial upper bound of $N_0 N_1$. The second lemma identifies when points can be solved for, possibly up to a reflection symmetry.

Lemma 3.2.4. *Let $\alpha \in \mathbb{Z}$. Then, equation $\rho_x = \alpha$ has at most $2(N_0 + N_1)$ integral solutions for x.*

Proof. We have that $\rho_x = x_0 x_1 (N_0 + 1 - x_0)(N_1 + 1 - x_1)$. For each $\beta \in [N_0]$, when we set

$x_0 = \beta$ we get: $\alpha = \beta x_1(N_1 + 1 - x_1)(N_0 + 1 - \beta) \implies$

$$\boxed{x_1^2 - (N_1 + 1)x_1 + \frac{\alpha}{\beta(N_0 + 1 - \beta)} = 0} \tag{3.4}$$

Solving the above quadratic equation (with real coefficients) for x_1, we get at most two integer solutions. For each $\gamma \in [N_1]$, when we set $x_1 = \gamma$ we get $\alpha = x_0\gamma(N_1+1-\gamma)(N_0+1-x_0)$ which implies

$$\boxed{x_0^2 - (N_0 + 1)x_0 + \frac{\alpha}{\gamma(N_1 + 1 - \gamma)} = 0} \tag{3.5}$$

We obtain $2N_1$ solutions by setting x_1 to each value in $[N_1]$ and solving for x_0. Thus, we obtain at most $2(N_0 + N_1)$ solutions for x.

□

We illustrate Lemma 3.2.5 in Figure 3.4.

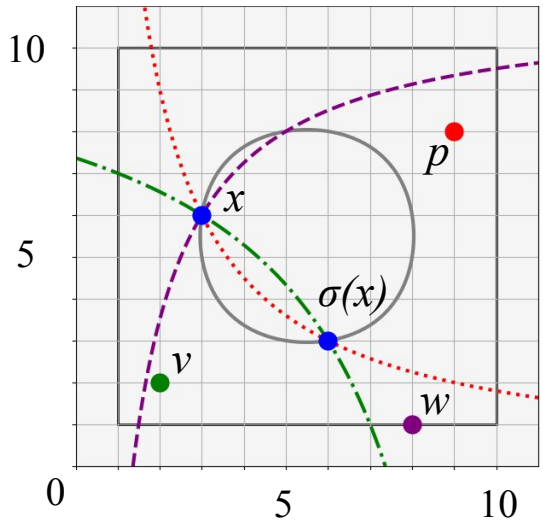


Figure 3.4: In Lemma 3.2.5, we know points v and w, and want to determine point x. The grey (solid), green (dashed-dotted) and purple (dashed) lines denote curves ρ_x, $\rho_{v,x}$ and $\rho_{w,x}$, respectively. The intersection of these three curves returns a unique location for x. To demonstrate that we need points in dominance (v) and anti-dominance (w) relationships with x, we also show that if we know some point p such that $x \preceq p$, $\rho_{p,x}$ in red (dotted) just gives us the same solutions as $\rho_{v,x}$.

Lemma 3.2.5. *Let $v, w \in [N_0] \times [N_1]$ and let $\alpha, \beta, \gamma \in \mathbb{Z}$. Then the system of equations*

$$\begin{aligned} \rho_x &= \alpha \\ \rho_{v,x} &= \beta \\ v &\preceq x \end{aligned} \tag{3.6}$$

has at most two integral solutions for x. If $\hat{x}$ is a solution, then $\sigma(\hat{x})$ is the other solution if and only if $v \preceq \sigma(\hat{x})$. Additionally, if we have

$$\begin{aligned} \rho_{w,x} &= \gamma \\ w &\preceq_a x \end{aligned} \tag{3.7}$$

then the system has at most one integral solution. Similarly, the system has at most one solution if the last equation of Systems (3.6) and (3.7) are replaced by $x \preceq_a v$ and $x \preceq w$, respectively.

Proof. Since $v \preceq x$, we know which coordinates of $\{v, x\}$ are minimal and maximal. Applying the formula for ρ, we write the first two equations of System 3.6 as

$$\begin{aligned} \alpha = \rho_x &= x_0 x_1 (N_0 + 1 - x_0)(N_1 + 1 - x_1) \\ \beta = \rho_{v,x} &= v_0 v_1 (N_0 + 1 - x_0)(N_1 + 1 - x_1) \end{aligned}$$

We then rewrite the above equations as

$$\boxed{\begin{aligned} \frac{\beta}{v_0 v_1} &= (N_0 + 1 - x_0)(N_1 + 1 - x_1) \\ \mu &= \alpha \frac{v_0 v_1}{\beta} = x_0 x_1 \\ \omega &= -\frac{\beta}{v_0 v_1} + N_0 N_1 + N_0 + N_1 + 1 + \mu \\ &= (N_1 + 1)x_0 + (N_0 + 1)x_1. \end{aligned}} \tag{3.8}$$

Using substitution and the quadratic formula we can obtain the following two solutions:

$$\begin{aligned}\hat{x}' &= \left(\frac{\omega - \mathsf{sqrt}}{2(N_1+1)},\ \frac{\omega + \mathsf{sqrt}}{2(N_0+1)}\right) \\ \hat{x}'' &= \left(\frac{\omega + \mathsf{sqrt}}{2(N_1+1)},\ \frac{\omega - \mathsf{sqrt}}{2(N_0+1)}\right)\end{aligned} \tag{3.9}$$

where $\mathsf{sqrt} = \sqrt{\omega^2 - 4\mu(N_1+1)(N_0+1)}$. Note that $\hat{x}'$ and $\hat{x}''$ are reflections across the main diagonal. Moreover, note that by Lemmas 3.2.2 and 3.2.3 we know that $\rho_x = \rho_{\sigma(x)}$ and $\rho_{v,x} = \rho_{v,\sigma(x)}$. Thus, both x' and x'' solve System (3.6) when they satisfy the third equation. For the backward direction, suppose that $\hat{x}$ is a solution and that $v \not\preceq \sigma(\hat{x})$. Then the third equation of System (3.6) would not be satisfied and the lemma follows.

Let us now consider the additional equations in System (3.7). Applying the formula for ρ given that $w \preceq_a x$ yields

$$\gamma = \rho_{w,x} = x_0 w_1 (N_0 + 1 - w_0)(N_1 + 1 - x_1).$$

We can then rearrange to obtain the system of equations

$$\boxed{\begin{aligned}\frac{\gamma}{(N_0+1-w_0)w_1} &= N_1 x_0 + x_0 - x_0 x_1 \\ -\frac{\beta}{v_0 v_1} + N_0 N_1 + N_0 + N_1 + 1 &= N_1 x_0 - N_0 x_1 + x_0 x_1\end{aligned}} \tag{3.10}$$

and then solve simultaneously to get unique values for x_0 and x_1.

Lastly, we consider the case when the last equation in Systems (3.6) and (3.7) are replaced with $x \preceq_a v$ and $x \preceq w$, respectively. By applying the rho equations we see that

$$\begin{aligned}\beta &= \rho_{v,x} = v_0 x_1 (N_0 + 1 - x_0)(N_0 + 1 - v_0) \\ \gamma &= \rho_{w,x} = x_0 x_1 (N_0 + 1 - w_0)(N_1 + 1 - w_1)\end{aligned}$$

which we can rearrange to get the system of equations

$$\boxed{\begin{aligned} \frac{\gamma}{(N_0+1-w_0)(N_1+1-w_1)} &= x_0x_1 \\ \frac{\beta}{v_0(N_1+1-v_1)} &= N_0x_1 + x_1 - x_0x_1. \end{aligned}} \tag{3.11}$$

and then solve for unique values of x_0 and x_1. □

We make the standard assumption that arithmetic on numbers of size (number of bits) $O(\log N_0N_1)$ can be done in constant time. We also employ full-precision arithmetic and use symbolic representations for non-integer values.

3.2.2 Technical Overview

In the remainder of this section, we work with a square domain $\mathcal{D} = [N] \times [N]$ in order to provide an overview of our work without the complications of a general domain. To understand the implications of solving the FDR problem, we are interested in the structure of $\mathsf{E}(D)$. A first observation is that applying the 8 "rigid motions of the square" (rotations and horizontal/vertical reflections) to D will result in equivalent databases. It is natural to conjecture that $\mathsf{E}(D)$ is generated this way, and that $|\mathsf{E}(D)| \leq 8$ (some databases will be invariant under these symmetries, resulting in an upper bound). Interestingly, the correct bound is exponential.

An initial attempt. Let us examine what happens if we naively generalize the prior attack of [70] attack to two dimensions. The first step is to use $\mathsf{RM}(D)$ to compute the query density ρ_x for every point x. Next, we can attempt to solve for x, up to the rigid motions of the square. This is depicted in Figure 3.5. As a function of two unknown coordinates over the reals, ρ_x is a degree-4 curve. Solving this involves intersecting the curve with the plane defined by α, which results in the curve on right side of the figure. It is already apparent that the situation in two dimensions is dramatically different form one dimension: Instead of getting two real points in this intersection, we get an infinite number of real solutions.

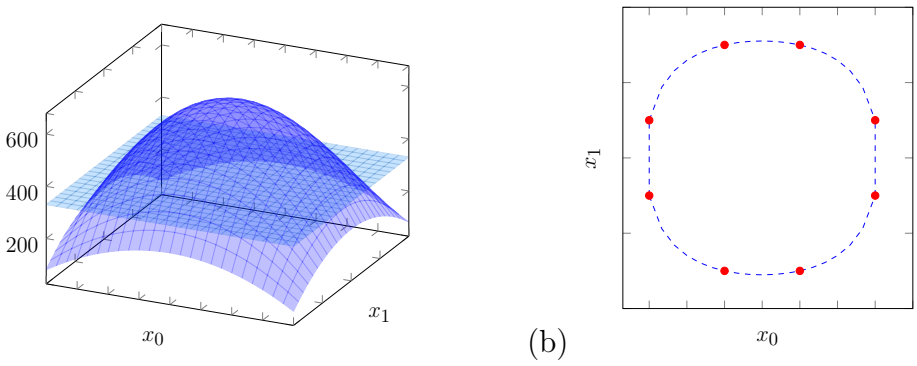


Figure 3.5: Solving $\rho_x = \alpha$ (Equation 3.2): (a) intersecting plane $z = \alpha$ with surface $z = x_0x_1(N+1-x_0)(N+1-x_1)$ defining ρ_x; (b) curve of the solutions, where the 8 integral points (in red) are symmetric with respect to rigid motions of the square. In general, there are additional integral points on this curve.

We partially resolve this situation by noting that the points we want on the curve must be ***integral*** since the record values in $\mathcal{D}$ take on integer values. One could potentially apply techniques from number theory to compute integral solutions directly, but this is beyond the scope of this work. By inspection, we can see that if $x \in \mathcal{D}$ is an integral solution, then we have up to eight integral solutions obtained by reflecting and rotating, as shown in Figure 3.5(b). These points are essentially unique, as the entire database can be permuted this way and be equivalent. But there is no reason that these should be the only integral solutions. Experiments indicate that the curve of Figure 3.5 can have an unbounded number of integral solutions (i.e. the number of solutions grows with N), partitioned into groups of at most 8 by the rigid motions.

Unfortunately for the attacker, another symmetry may occur that is not covered by the rigid motions of the square. For example, consider the database of Figure 3.6, which comprises 8 red points. Reflecting any subset of the points results in an equivalent database. Moreover, these reflections are not rigid motions of the square.

FDR in two dimensions: Our approach. We obtain an FDR algorithm by giving a new approach that teases apart the subtle structure of $\mathsf{E}(D)$, even when it is exponentially large. The first step is to identify $2, 3$, or 4 extreme points that "contain" the rest of the database;

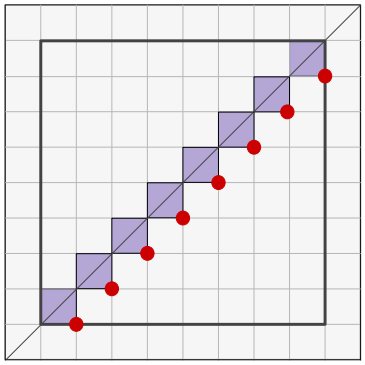


Figure 3.6: Reflecting any subset of the 8 database points yields an equivalent database. Further applying rigid motions, we get a total of $8 \times 2^8 = 1,024$ equivalent databases.

in particular these points will collectively achieve the maximum and minimum values in each dimension. We find these by looking for a minimal set of points such that their co-occurrence in a query implies the entire database is in that query.

We then algebraically solve for the possible assignments of these points in $\mathcal{D}$ by carefully applying Lemmas 3.2.4 and 3.2.5, obtaining a polynomial list of solutions. Then for each possible solution, we recover a family of equivalent databases, organized by their freely-moving "components". Taking a union over all of the families for the possible solutions for the extreme points gives $\mathsf{E}(D)$.

3.3 Classifying Equivalent Databases

Before delving into the algorithm, we need to know what the best we can do is. Given $\mathsf{RM}(D)$ for some database D, an algorithm can at best find $\mathsf{E}(D)$, the set of all databases D' such that $\mathsf{RM}(D') = \mathsf{RM}(D)$. Unlike the one-dimensional case, $\mathsf{E}(D)$ has more structure than a simple reflection. As shown in Figure 3.6, we can obtain databases equivalent to D by repeatedly performing a transformation that replaces a subset of points with their reflections without affecting the dominance relation of these points with respect to all the other points.

Components. To understand equivalent databases, we must introduce some terminology.

Definition 11. *A **component** of D is a minimal non-empty subset C of points of D such that for every point $p \in C$ and point $q \in D$ such that $q \notin C$, one of the following holds (see Figure 3.7):*

- *p and $\sigma(p)$ both dominate q; or*
- *p and $\sigma(p)$ are both dominated by q.*

It is immediate that any two components of a database are disjoint, because if their intersection was non-empty, it would form a smaller component. The components of a database are uniquely determined, as formally stated below.

Lemma 3.3.1. *Any database can be uniquely partitioned into components.*

Proof. Suppose P_1 and P_2 are two distinct partitions of a database into components. Then, their components can be ordered by domination, going from bottom to top along the diagonal. Let C_1 and C_2 be the first components of P_1 and P_2 that differ. Thus for every $p \in C_1$ and q not in C_1 or an earlier component, p and $\sigma(p)$ are dominated by q. The same holds for C_2.

Assume without loss of generality that there is point $p \in C_1$ such that $p \notin C_2$. If C_2 is not a subset of C_1, then there is some point $q \in C_2$ and $q \notin C_1$. Partition P_1 indicates that $p, \sigma(p) \preceq q$ (since q is not in an earlier component or in C_1), and partition P_2 similarly indicates that $q, \sigma(q) \preceq p$. These imply $p = q$, a contradiction. If C_2 is a strict subset of C_1, then it contradicts the minimality of C_1, as it is a smaller component contained in C_1. Thus $C_1 = C_2$, and the partitions must be the same. □

A point p of a domain $\mathcal{D}$ is said to be ***reflectable*** if the reflection $\sigma(p)$ of p is a point of $\mathcal{D}$. We extend this definition to components by saying that a component C of a database D is reflectable if all the points of C are reflectable. For technical reasons, if a component consists of a single point on the main diagonal, then we define it to not be reflectable. The following lemma, illustrated in Figure 3.7, states that replacing the points of a reflectable component with their reflections leaves the search pattern unchanged.

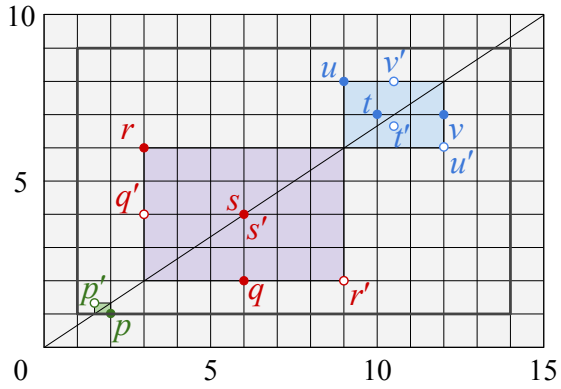


Figure 3.7: A database D over domain $[14] \times [9]$. Solid circles represent database points and hollow circles represent their reflections. D has components $C_1 = \{p\}$, $C_2 = \{q, r, s\}$, and $C_3 = \{t, u, v\}$. C_1 and C_3 are nonreflectable while C_2 is reflectable. Replacing the points of C_2 with their reflections yields a database equivalent to D.

Lemma 3.3.2. *Let C be a reflectable component of a database, D, and let D' be database obtained from D by replacing C with component C' comprising $\sigma(p)$ for every point $p \in C$. We have that D and D' are equivalent with respect to the response multiset, i.e.,* $\mathsf{RM}(D) = \mathsf{RM}(D')$.

Proof. We give here the proof for the case when D is over a square domain, i.e., a domain $\mathcal{D} = [N_0] \times [N_1]$ such that $N_0 = N_1$. This case is easier to deal with since the reflection of every point of $\mathcal{D}$ is also in $\mathcal{D}$. The proof for a general domain has a similar structure but involves additional details.

We show that D and D' are equivalent by defining a one-to-one mapping between queries on D and queries on D' such that queries mapped to each other have the same response. Namely, given a query q for D with access response $\mathsf{Resp}(D, q)$, we generate a query q' for D', such that $\mathsf{Resp}(D, q) = \mathsf{Resp}(D', q')$.

Let B the union of the components of D preceding C on the diagonal. Also, let A be the union of the components of D following C on the diagonal. We have that D' consists of B followed by C', followed by A (see Figure 3.8). We consider five cases:

1. $\mathsf{Resp}(D, q)$ contains no points in C: We map query q to itself as D and D' are identical but for the points in C.

2. $\mathsf{Resp}(D, q)$ contains only points in C: We map query $q = (c, d)$ to $q' = (\sigma(c), \sigma(d))$. We have that if $c \preceq p \preceq d$, then $\sigma(c) \preceq \sigma(p) \preceq \sigma(d)$. Thus, $\mathsf{Resp}(D, q) = \mathsf{Resp}(D', q')$.

3. $\mathsf{Resp}(D, q)$ has points in B and C but not in A. We map query $q = (c, e)$ to $q' = (c, e')$, where $e' = \sigma(e)$ (see Figure 3.8). Let range (c, f) be the intersection of ranges (c, e) and (c, e'). Also, let ranges (d, e) and (d', e') be the remaining parts of (c, e) and (c, e'), respectively. Since the reflections of the points of C in (c, f) are also in (c, f), we have that (c, f) contains the same points of C and C'. Consider now the points of C in (d, e). This is the scenario of Case 2 above and thus we have that these points are the same as the points of C' in (d, e'). We conclude that $\mathsf{Resp}(D, q) = \mathsf{Resp}(D', q')$.

4. $\mathsf{Resp}(D, q)$ has points in A and C but not in B. This case is symmetric to Case 3 and can be proved similarly.

5. $\mathsf{Resp}(D, q)$ contains points in A and B. Here, all points of C and C' are contained in $\mathsf{Resp}(D, q)$. Thus, we map q to itself.

We can similarly show that given a query q' for D', we can generate query q for D, such that $\mathsf{Resp}(D', q') = \mathsf{Resp}(D, q)$. Also, this mapping is the inverse of the previous one. It follows that $\mathsf{RS}(D) = \mathsf{RS}(D')$ and thus D and D' are equivalent. □

The main argument of this proof is schematically illustrated in Figure 3.8.

Before going further, we show that the set of a equivalent databases may be arbitrarily large (in contrast to the one-dimensional case). We can exploit Lemma 3.3.2 to build a database that admits an exponential number of equivalent databases (see Figure 3.6).

Theorem 3.3.3. *For every integer R, there exists a family of 2^R databases with R points that are equivalent to each other.*

Proof. Let D be the database over domain $[R+1]\times[R+1]$ with points $(i+1, i)$ for $i = 1, \cdots, R$. We have that D has R reflectable components, each comprising a single point. Applying

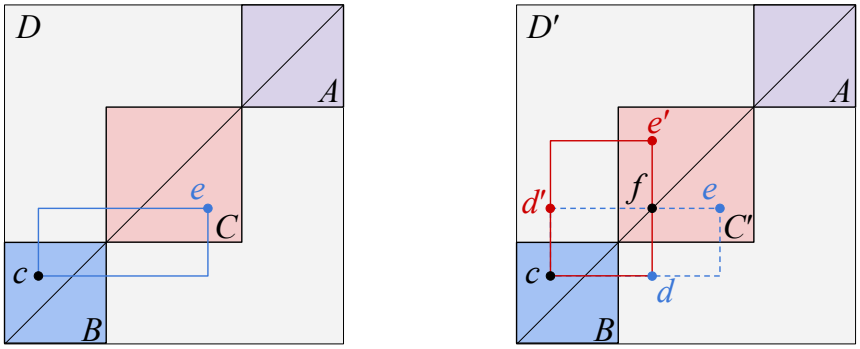


Figure 3.8: Case 3 of the proof of Lemma 3.3.2: Database D' is obtained from D by reflecting component C to yield C'; the blue range query (c, e) on D and the red range query (c, e') on D' return the same response, as seen from the intersection (c, f) and differences, (d, e) and (d', e'), of the two ranges.

Lemma 3.3.2, we obtain 2^R equivalent databases by replacing subsets of the points of D with their reflections. □

Classification theorem. We now prove that for any database D, the set of equivalent databases can be systematically described. At a high level, we show that any database equivalent to D can be formed by starting from a small number of "seed" databases and reflecting their components.

Theorem 3.3.4. *Given a database D with points from domain $[N_0] \times [N_1]$, there exists a set $\mathcal{S}$ of $O(N_0 + N_1)$ databases such that any database equivalent to D can be obtained from a database $D' \in \mathcal{S}$ by reflecting a subset of the reflectable components of D', and then reflecting the resulting database vertically and/or horizontally.*

The proof includes much of the reasoning used to prove our later algorithm correct, but we present a self-contained version for clarity. The seed databases correspond to sets of integral solutions to certain equations, which are theoretically possible up to the stated bound. In experiments with real data, we only ever needed one seed database. We did observe real databases with several reflectable components.

Proof. Fix a database $D \in \mathcal{D}^R$. We start by observing that D contains a set $\{\mathit{left}, \mathit{right}, \mathit{bot}, \mathit{top}\}$ of 2 to 4 extreme points that achieve the minimum and maximum each dimension (see Figure 3.9). By reflecting D vertically and/or horizontally (operations which preserve equivalence), we may assume that *right* dominates *left*.

In any database that is equivalent to D, the extreme points must have the same query densities as in D. One of these four extreme points must achieve minimal value in the first coordinate. For each of those choices, by Lemma 3.2.4, there are $2(N_0 + N_1)$ solutions for *left* that appear in an equivalent database. For each of these solutions, by the first part of Lemma 3.2.5, there are at most two solutions for *right* (using the assumption that $\mathit{left} \preceq \mathit{right}$). Thus there are at most $O(N_0 + N_1)$ possible values for *left* and *right* in any equivalent database with $\mathit{left} \preceq \mathit{right}$.

For each of these possible solutions for $\mathit{left}, \mathit{right}$, there may or may not exist a database D' that has extreme points set to those solutions and is equivalent to D. If there is none, we discard those solutions. Otherwise, we take D' to be any such database and add it to $\mathcal{S}$. As above, we have assumed that *left* is dominated by *right* in D'.

We now argue that any database $\hat{D}$ that is (1) equivalent to D (and hence D') and (2) has the same $\mathit{left}, \mathit{right}$ as D', can be obtained from D' by diagonally reflecting components of D'. By Lemma 3.3.2, all of the databases obtained in this way will be equivalent to D', so this will prove the theorem.

Every point in $\hat{D}$ lies in the regions S_1, S_2, or S_3 depicted in Figure 3.10, and moreover must lie in the same region as it does in D' (otherwise $\hat{D}$ is not equivalent). Any point of $\hat{D}$ in S_1 or S_3 uniquely determined by query densities in D', by the second part of Lemma 3.2.5 (taking $v = \mathit{left}, w = \mathit{right}$ and x as the unknown point in S_1 or S_3, and applying the two versions of the second part to S_1 and S_3 respectively). Moreover, by the first part of Lemma 3.2.5, every point in S_2 is determined up to reflection by σ, and one of those solutions must be the corresponding point in D' (this takes $v = \mathit{left}$ in the lemma).

We next observe that in $\hat{D}$, how the points are divided into the components is the same as in D', even though the points in S_2 may not be uniquely determined. This is because reflecting any point of a database by σ does not change which points are in which components.

Finally, we show that $\hat{D}$ can be obtained by reflecting components of D' contained in S_2. Fix a component C of $\hat{D}$, and order its non-diagonal points $\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_k$ by their low projections onto the diagonal (the diagonal points automatically match because they only have one solution). Let $u'_1, \ldots, u'_k$ be the corresponding points of D'. The point u'_1 is reflectable since C is reflectable. We also know that u'_1 does not lie on the diagonal, because otherwise u'_1 could be removed from C to form a smaller component, contradicting the minimality in the definition of C (further points may however lie on the diagonal). Since $\hat{u}_1$ was determined up to reflection, it is either u'_1 or $\sigma(u'_1)$, which are distinct. If $\hat{u}_1 = u'_1$, we claim the entire component of $\hat{D}$ matches the component in D' ($\hat{u}_i = u'_i$ for all i). Otherwise, we claim that $\hat{u}_i = \sigma(u'_i)$ for all i.

Now suppose $\hat{u}_1 = u'_1$; the case $\hat{u}_1 = \sigma(u'_1)$ is similar. We claim that $\hat{u}_2 = u'_2$. The key observation is that, by our ordering of the points, u'_1 and u'_2 must fall into one of the following relationships:

- $u'_1 \preceq u'_2$ and $u'_1 \preceq \sigma(u'_2)$
- $u'_1 \preceq u'_2$ and $u'_1 \preceq_a \sigma(u'_2)$
- $u'_1 \preceq_a u'_2$ and $u'_1 \preceq \sigma(u'_2)$
- $u'_1 \preceq_a u'_2$ and $u'_1 \preceq_a \sigma(u'_2)$.

The first case cannot happen, because then u'_1 could be removed from the component, contradicting minimality. (If u'_1 dominates both of these points, then it also dominates $u'_i, \sigma(u'_i)$ for all i).

We next address the second case, and claim that $\hat{u}_2$ must equal u'_2 in order for $\hat{D}$ to be equivalent to D'. (The third case is similar.) If $\hat{u}_2$ were equal to $\sigma(u'_2)$ instead of, then

there would exist a query over $\hat{D}$ containing *left* and $\hat{u}_2$ and not $\hat{u}_1$. But in D' all queries containing *left* and u'_2 also contain u'_1, so the databases would not be equivalent.

Finally, in the fourth case, we have $u'_1 \preceq_a u'_2$, so by Lemma 3.2.5 (with $w = \mathit{left}, x = u'_1$), u'_2 is uniquely determined, and we must have $\hat{u}_2 = u'_2$ in order for the databases to be equivalent.

This shows that $\hat{u}_2 = u'_2$. We can continue the argument for the rest of the $\hat{u}_i$. In place of u'_1, we find some u'_j $(j < i)$ that is related in one of the latter three above ways to u'_i that u'_1 was related to u'_2. Such an j must exist, because otherwise we would have that $u'_j \preceq u'_i, u'_j \preceq u'_i$ for all $j < i$, and we could remove u'_i and the subsequent points to form a smaller component. (Note that j might not be $i - 1$.) This completes the claim that the component either matches or is entirely reflected. This argument also shows that non-reflectable components of D' must be equal in $\hat{D}$. This completes the proof. □

3.4 Full Database Reconstruction

In this section, we present our full database reconstruction (FDR) attack in two dimensions.

3.4.1 Overview of the attack

Our attack relies on the contents of $\mathsf{RS}(D)$ and $\mathsf{RM}(D)$ and comprises the following steps:

1. Identify the extreme points of the database, including any corner points. (Algorithm 3)
2. Extract the left-most, *left*, and right-most, *right*, points and segment the remaining points in three sets: one with all points above left and right, one with all points between them, and one with all points below them. (Algorithm 4)
3. Find all possible locations for points *left* and *right*, and use them to identify one or two locations for every point in the database. (Algorithm 5)

4. Using the recovered locations, partition the database into components. (Algorithm 6)

5. Prune the locations for the points in each partition down to one location per point. (Algorithm 7)

3.4.2 Preprocessing

Before we delve into the algorithm, we will preprocess the input. Given multiset $\mathsf{RM}(D)$, we generate the corresponding set $\mathsf{RS}(D)$. At this point, we would like to note that $\mathsf{RM}(D)$ has size $O((N_0N_1)^2) = O(N^2)$ and $\mathsf{RS}(D)$ has size $O(\min(R^4, N^2))$. A single response can contain up to R identifiers. Thus, it takes time $O(\min(R^5, RN^2))$ to read $\mathsf{RS}(D)$ and time $O(RN^2)$ to read $\mathsf{RM}(D)$.

We also preprocess $\mathsf{RM}(D)$ and $\mathsf{RS}(D)$ making sure that each value in the domain corresponds to at most one identifier. We do that by finding the smallest set S in $\mathsf{RS}(D)$ that contains a given ID. Then, we go through $\mathsf{RM}(D)$ and $\mathsf{RS}(D)$ replacing set S from each response with a new identifier.

3.4.3 Get extremes

The first step of the reconstruction algorithm finds a minimal set of extreme points of database D, i.e., a set $E \subseteq D$ of smallest size such that for any point $p \in D$ there exist points *left*, *right*, *bot*, and *top* in E such that $left_0 \leq p_0 \leq right_0$ and $bot_1 \leq p_1 \leq top_1$. Note that the same point of E may be the minimum or maximum in both dimensions, i.e., we may have $left = bot$ or $right = top$. We call such a point a ***corner*** of the database.

Suppose database D has at least two distinct points. We consider the following three cases for a minimal set of extreme points, E, of D (see Figure 3.9):

Case 1: E has two points, both corners: $p = left = bot$ and $q = right = top$.

Case 2: E has three points, one of which is a corner: $p = left$, $q = right = top$, and $r = bot$.

Case 3: E has four points, none of which is a corner: $p = left$, $q = top$, $r = right$, and $s = bot$.

Algorithm 3 takes as input $\mathsf{RS}(D)$ and returns a constant size list of hashmaps. Each hashmap contains four entries, each corresponding to an extreme point (minimum or maximum coordinate) in a dimension.

We first identify the points on two, three or four edges of the database, depending on if we have case (1), (2) or (3) respectively. We do so by finding the second largest response in $\mathsf{RS}(D)$, S_1. The difference of S_1 with the set containing all database points is an edge of the database. We similarly find the rest of the edges.

Then, we identify the extreme points of each edge. The key idea is that each extreme point only has one point right next to it on the edge. The non-extreme points have a point on either side. Thus, if we look at sets of size two in $\mathsf{RS}(D)$ that contain only points in one edge, the extreme points will each be in exactly one such set. Every other point will be in two.

Once we have identified the extreme points of each edge, we need to find any corners, and decide on which point from each edge we'll return as extreme. We simply iterate through all subsets of size 2,3, and 4 of extreme points and pick the first subset, such that the smallest response in $\mathsf{RS}(D)$ that contains all points in this subset is the largest one. The algorithm then returns all possible configurations of this subset of points.

Lemma 3.4.1. *Let D be a database with R records and let $\mathsf{RS}(D)$ be its response set. Algorithm 3 returns all possible configurations of extreme points of D (up to symmetries) in time $O(\min(R^5, RN^2))$.*

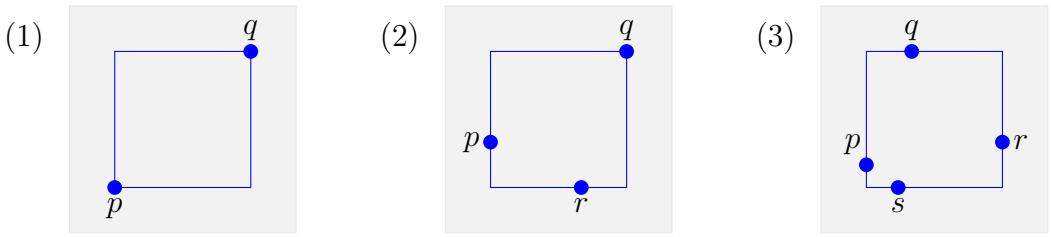


Figure 3.9: Three mutually exclusive cases for a minimal set of extreme points of a database: (1) two corners, p and q; (2) one corner, q; (3) no corners.

Algorithm 3 FindExtremes(RS(D))

1: Let L be the largest set in RS(D) and Edges, PosExtremes empty lists
2: Let S_1 be the 2^{nd} largest set in RS(D). Add $L - S_1$ to Edges.
3: Let S_2 be the 2^{nd} largest set containing $L - S_1$. Add $L - S_2$ to Edges.
4: Let S_3 be the 2^{nd} largest set containing $L - S_1$ and $L - S_2$ (if it exists). Add $L - S_3$ to Edges.
5: Let S_4 be the 2^{nd} largest set containing $L - S_1$, $L - S_2$ and $L - S_3$ (if it exists). Add $L - S_4$ to Edges.
6: Let PosExtremes be an empty list.
7: **for all** $E \in$ Edges **do**
8: **if** $|E| = 1$ **then**
9: Add the one ID in E to PosExtremes
10: **else**
11: Consider the responses of size 2 in RS(D) containing only IDs of E.
12: Count in how many such responses each ID of E appears.
13: Add any IDs that appear exactly once to PosExtremes
14: **for** $i = 2$ **to** 4 **do**
15: **for all** subsets of IDs $E \subseteq$ PosExtremes such that $|E| = i$ **do**
16: **if** L is the only set in RS(D) containing E **then**
17: // The i IDs of E refer to a minimal set of extreme points
18: Let *PosConfigs* be an empty list
19: **for all** possible configurations of extreme point IDs in E **do**
20: Define map mapping keys left, right, bot, and top, to respective IDs of E.
21: Add this map to *PosConfigs*.
22: **Return** *PosConfigs*

Proof. First we argue that after line 5 the set Edges must contain the extreme points. The second largest query in RS(D) must exclude at least one extreme point p. Suppose for a contradiction that p is not extreme, then we could extend the query to include p which would thus result in a strictly larger query that still doesn't contain all records. Now consider the second largest query that contains p. Once again, the remaining point(s) must be extreme in another direction. If not, then we could extend that query to include the non-extreme point, which would result in a strictly larger query that is not the whole database. Since we repeat this four times, each time ensuring that the previously recovered extreme points are included in the second largest query, then we are able to recover all extreme points (not necessarily a minimal set).

The loop on line 7 then checks if there are any potential corner points. Note that any set of IDs in $L - S_i$ (for $i = 1, \ldots, 4$) added to Edges must correspond to points with the same value in the extreme dimension and different values in the other. If there is only one point in

this set, then we must add it to PosExtremes.

Else, we need to locate any possible corners on this edge. For example, in case (2), we only ever see three edges of the database, but we still need a point on each of the four edges. In order to do so with only three edges, we need to locate the corners.

To find potential corners, if there are multiple IDs in this edge we select the points that only appear in one set in $\mathsf{RS}(D)$ that is of size 2 and restricted to points in that edge. Note that any point that is a corner must satisfy this condition. At the end of this for loop, PosExtremes will therefore contain any corner points.

Lastly, we will show that a subset $\mathcal{S} \subseteq [R]$ is a valid set of extreme point identifiers iff the minimal query that contains those points must also contain the whole database. Let $\mathit{left}, \mathit{right}, \mathit{bot}, \mathit{top} \in D$ be the set of extreme points (not necessarily unique) that achieve minimum and maximum values in the first and second coordinates, respectively. Then for all $p \in D$, we have $\mathit{left}_0 \leq p_0 \leq \mathit{right}_0$ and $\mathit{bot}_1 \leq p_1 \leq \mathit{top}_1$. Let $q = ((\mathit{left}_0, \mathit{bot}_1), (\mathit{right}_0, \mathit{top}_1))$. Then, by definition, the query q returns all $p \in D$ such that

$$(\mathit{left}_0, \mathit{bot}_1) \preceq p \preceq (\mathit{right}_0, \mathit{top}_1)$$

i.e. all of D. For the backward direction, suppose that the minimal query containing $\mathit{left}, \mathit{right}, \mathit{bot}, \mathit{top} \in D$ also contains the whole database. Suppose for a contradiction that one of these points is not extreme and so there exists, WLOG, $p \in D$ be such that

$$p_0 < \min(\mathit{left}_0, \mathit{right}_0, \mathit{bot}_0, \mathit{top}_0).$$

But that means that $(\mathit{left}_0, \mathit{bot}_1) \not\preceq p$ and hence p cannot be in the minimal query containing those four points, which is a contradiction. Hence an element of $\{\mathit{left}_0, \mathit{right}_0, \mathit{bot}_0, \mathit{top}_0\}$ must achieve the minimal value along the first coordinate. A similar argument can be made for the other extremes.

Searching through $\mathsf{RS}(D)$ to find the second largest queries in lines 1 to 5 will take time $O(\mathsf{RS}(D))$. In the worst case, $|\mathsf{Edges}| = R$ and then finding a set satisfying the else statement takes time $O(|\mathsf{RS}(D)|)$. Thus, the for loop on line 7, takes $O(R|\mathsf{RS}(D)|)$ time. Since $\mathsf{PosExtremes}$ only contains points that are singular in an edge or potential corners, then $|\mathsf{PosExtremes}| \leq 8$, which implies that the for loops on lines 14 and 15 exhibit a constant number of iterations. Checking that L is the only set in $\mathsf{RS}(D)$ containing E can be done with a number of element membership searches linear in $|\mathsf{RS}(D)|$. Also, checking all possible configurations contributes a constant factor. Hence, overall Algorithm 3 runs in time $O(R|\mathsf{RS}(D)|) = O(\min(R^5, RN^2))$. □

3.4.4 Segment the database

Given a set of extreme points *left*, *right*, *bot*, and *top* computed by Algorithm 3, our goal is to segment the points of the database D into three sets, S_1, S_2, and S_3 according to their vertical (second coordinate) arrangement with respect to the extreme points. Namely, these sets are defined as follows:

- S_1 comprises the points of D that are vertically above both *left* and *right*;
- S_2 comprises the points of D that are vertically in between *left* and *right* (included);
- S_3 comprises the points of D that are vertically below both *left* and *right*.

Note that in Case 1 (as defined in Section 3.4.3), we have a trivial segmentation where $S_1 = S_3 = \emptyset$ and $S_2 = D$. Two subcases for the segmentation in Case 3 are illustrated in Figure 3.10.

Algorithm 4 (*Segmentation*) takes as input $\mathsf{RS}(D)$ and *PosConfigs* (output by Algorithm 3) and returns a list of tuples, where each tuple comprises of two IDs, for *left* and *right*, and three sets of IDs.

Algorithm 4 *Segmentation*$(PosConfigs, \mathsf{RS}(D))$

1: Let *Segmentations* be an empty list.
2: **for all** hashmaps $H \in PosConfigs$ **do**
3: Let S_2 be the smallest set in $\mathsf{RS}(D)$ containing $H[\mathsf{left}]$ and $H[\mathsf{right}]$
4: Let T be the smallest set in $\mathsf{RS}(D)$ containing $H[\mathsf{top}]$, $H[\mathsf{left}]$ and $H[\mathsf{right}]$
5: $S_1 = T - S_2$
6: $S_3 = D - T$
7: Add $((H[\mathsf{left}], H[\mathsf{right}]), (S_1, S_2, S_3))$ to *Segmentations*.
8: **Return** *Segmentations*

Lemma 3.4.2. *Let $D \in \mathcal{D}^R$ be a database with R records and let $\mathsf{RS}(D)$ be the response set of D. Let PosConfigs be the output of Algorithm 3 on $\mathsf{RS}(D)$. Then Algorithm 4 returns a list that includes each possible set of extreme points, denoted $(left, right)$, and their corresponding database segments (S_1, S_2, S_3) in $O(\min(R^5, RN^2))$ time.*

Proof. Note that for all $p \in D$, $left_0 \leq p_0 \leq right_0$ and $bot_1 \leq p_1 \leq top_1$. By definition, a query $q = (c, d)$ returns all points p such that $c \preceq p \preceq d$. Moreover, the smallest query containing two points is the query defined by those two points.

S_2 is computed to be the smallest set in $\mathsf{RS}(D)$ containing *left* and *right*. Query $q = (left, right)$ must return all points $p \in D$ such that $left \preceq p \preceq right$, which precisely corresponds to all points of D vertically in between *left* and *right*, as desired.

S_1 is defined as $T - S_2$, where T is the smallest set in $\mathsf{RS}(D)$ containing *top*, *left* and *right*. In particular, T contains all points corresponds to the query $q = (left, (right_0, top_1))$ i.e. all $p \in D$ such that $left \preceq p \preceq (right_0, top_1)$. In particular, $T = S_1 \cup S_2$ and so $S_1 = T - S_2$. Moreover, $D = S_1 \cup S_2 \cup S_3$, thus $S_3 = D - T = D - (S_1 \cup S_2)$. The correctness of the segments follows.

The for loop on line 2 runs through a constant number of iterations, as there is a constant number of possible configurations for the extreme points. Finding the smallest sets in lines 3 and 4 is linear in $R|\mathsf{RS}(D)|$. Steps 5 and 6 are linear in the sizes of the sets, which is $O(R)$. The total runtime is $O(R|\mathsf{RS}(D)| + R) = O(\min(R^5, RN^2))$. □

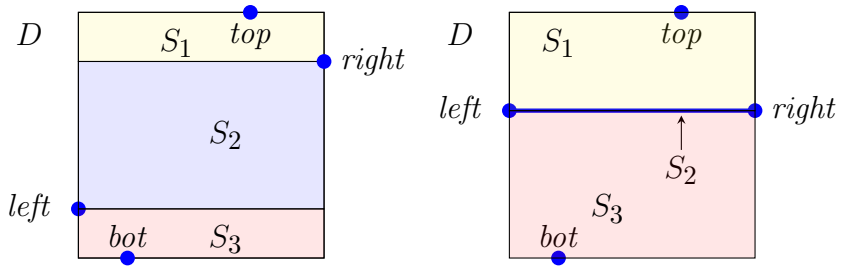


Figure 3.10: Segmenting the database into three sets of points.

3.4.5 Find candidate locations

Algorithm 5 finds candidate locations for all the database records. First we find the candidate locations for all the database records. Each hashmap in list *Segmentations* has two keys, denoted *left*, and *right*, which provide the IDs of the leftmost and rightmost points, as defined in Section 3.4.3. First, we use ρ_{left} to compute *left* such that $left_0 \in [N_0]$ and $left_1 \in [N_1]$. We can obtain at most $2(N_0 + N_1)$ solutions as shown in Lemma 3.2.4. We then use a second parameter, $\rho_{left,right}$, to obtain the set of equations in System 3.6, and then solve for all valid solutions of $left_0, left_1, right_0$, and $right_1$. Once we have located *left* and *right*, we can use these points to compute at most two potential solutions for the remaining records using Lemma 3.2.5.

Algorithm 5 *Solve*(*Segmentations*, $\mathsf{RS}(D)$)

Initialize lists *Extremes*, and *Solutions*; Initialize hashmap Rho
for $H \in PosConfigs$ **do**
    Let $\mathsf{ID}_1 = H[\mathsf{left}]$ and $\mathsf{ID}_2 = H[\mathsf{right}]$
    for all record $\mathsf{ID} \in D$ **do**
        Let m be the number of responses in $\mathsf{RS}(D)$ containing ID
        Let m_1 be the number of responses in $\mathsf{RS}(D)$ containing ID_1, ID
        Let m_2 be the number of responses in $\mathsf{RS}(D)$ containing ID_2, ID
        Let $\mathsf{Rho}[\mathsf{ID}] = m$, $\mathsf{Rho}[(\mathsf{ID}_1, \mathsf{ID})] = m_1$, $\mathsf{Rho}[(\mathsf{ID}_2, \mathsf{ID})] = m_2$.

// First, find all possible solutions for extreme points.
for all $((\mathsf{ID}_1, \mathsf{ID}_2), (S_1, S_2, S_3)) \in Segmentations$ **do**
    Initialize list L.
    for $left_0 = 1, \ldots, N_0$ **do**
        Compute solutions $left_1', left_1''$ of Eq. 3.4, $\beta = left_0$, $\alpha = \mathsf{Rho}[\mathsf{ID}_1]$.
        if $left_1' \in \mathcal{D}$ **then** add $(left_0, left_1')$ to L.
        if $left_1'' \in \mathcal{D}$ **then** add $(left_0, left_1'')$ to L.
    for $left_1 = 1, \ldots, N_1$ **do**
        Compute solutions $left_0', left_0''$ of Eq. 3.5, $\gamma = left_1$, $\alpha = \mathsf{Rho}[\mathsf{ID}_2]$.
        if $left_0' \in \mathcal{D}$ **then** add $(left_0', left_1)$ to L.
        if $left_0'' \in \mathcal{D}$ **then** add $(left_0'', left_1)$ to L.
    for all $left \in L$ **do**
        Compute solutions $right'$ and $right''$ using System (3.8) with $\alpha = \mathsf{Rho}[\mathsf{ID}_2]$, $\beta = \mathsf{Rho}[(\mathsf{ID}_1, \mathsf{ID}_2)]$, and $v = left$.
        if $right' \in \mathcal{D}$ **then** add $((\mathsf{ID}_1, \mathsf{ID}_2), (left, right'), (S_1, S_2, S_3))$ to *Extremes*.
        if $right'' \in \mathcal{D}$ **then** add $((\mathsf{ID}_1, \mathsf{ID}_2), (left, right''), (S_1, S_2, S_3))$ to *Extremes*.

// Now find at most two solutions for all other points.
for all $((\mathsf{ID}_1, \mathsf{ID}_2), (left, right), (S_1, S_2, S_3)) \in Extremes$ **do**
    Initialize hashmap H.
    for all $\mathsf{ID} \in S_1$ **do**
        Compute *ans* to System (3.10) with $\beta = \mathsf{Rho}[(\mathsf{ID}_1, \mathsf{ID})]$, $\gamma = \mathsf{Rho}[(\mathsf{ID}_2, \mathsf{ID})]$, $v = left$, $w = right$.
        if $ans \notin \mathcal{D}$ **then go to** line 27.
        else Let $H[\mathsf{ID}_2] = \{ans\}$.
    for all $\mathsf{ID} \in S_2$ **do**
        Compute ans, ans' using System (3.8) with $\alpha = \mathsf{Rho}[\mathsf{ID}]$, $\beta = \mathsf{Rho}[(\mathsf{ID}_1, \mathsf{ID})]$, and $v = left$.
        if $ans \in \mathcal{D}$ **then** Let $H[\mathsf{ID}] = \{ans\}$.
        if $ans' \in \mathcal{D}$ **then** Let $H[\mathsf{ID}] = H[\mathsf{ID}] \cup \{ans'\}$.
        if $(ans, ans') \notin \mathcal{D}^2$ **then go to** line 27.
    for all $\mathsf{ID} \in S_3$ **do**
        Compute *ans* to System (3.11) with $\beta = \mathsf{Rho}[(\mathsf{ID}_1, \mathsf{ID})]$, $\gamma = \mathsf{Rho}[(\mathsf{ID}_2, \mathsf{ID})]$, $v = left$, $w = right$.
        if $ans \notin \mathcal{D}$ **then go to** line 27.
        Let $H[\mathsf{ID}] = \{ans\}$.
    Add H to *Solutions*.
Return *Solutions*.

Lemma 3.4.3. *Let $D \in \mathcal{D}^R$ be a database with R records and let $\mathsf{RM}(D)$ be its response multiset. Algorithm Solve computes in $O(RN^2)$ time a list of hashmaps, denoted Solutions, such that for each database $\hat{D}$ equivalent to D, i.e., $\hat{D} \in \mathsf{E}(D)$, there exists $H \in$ Solutions with the property that for all $\mathsf{ID} \in [R]$, we have $\hat{D}[\mathsf{ID}] \in H[\mathsf{ID}]$.*

Proof. By Lemma 3.4.1 we know that Algorithm 3 correctly outputs the configurations of the extreme points, *PosConfigs*. The for loop on line 2 iterates through each hashmap in *PosConfigs*, computing the necessary query densities and storing them in a hashmap Rho which maps IDs to their corresponding query densities. The correctness of these values follows from the definition of the query density equations. For each configuration in *PosConfigs*, the segments are then computed and output as *Segmentations*. In lines 11 to 24, Algorithm 5 iterates through *Segmentations* and computes the at most $4(N_0 + N_1)$ possible values of *left* and *right*. The correctness of this step follows from Lemma 3.2.4. By Lemma 3.2.5 we can then solve each remaining point up to two values. Let $\hat{D}$ be an equivalent database to D. Any $p \in \hat{D}$ in Segment S_2, S_1 or S_3, must be consistent with either Systems (3.8), (3.10), or (3.11), respectively. Since Algorithm 5 evaluates the possible values for each point in the database given each pair of extreme points in *Segmentations*, then for any equivalent $\hat{D}$ there exists some $H \in$ *Solutions* such that for all identifiers $\mathsf{ID} \in [R]$, we have that $\hat{D}[\mathsf{ID}] \in H[\mathsf{ID}]$.

Computing the necessary query density functions takes time $O(RN^2)$. The loop on line 11 iterates a constant number of times and the three nested for loops iterate $O(N_0 + N_1)$ times. Checking that some element is in the domain and then adding it to a list takes time $O(1)$.

The loop starting at line 27 iterates through a list of size $O(N_0 + N_1)$. The three inner for loops run through all points in S_1, S_2, and S_3 which is at most R iterations. Initializing a hash map, evaluating all possible values, checking and then adding an element all take constant time. Thus, the total runtime of this part of the algorithm is $O(R(N_0 + N_1))$, which yields an overall runtime of $O(RN^2)$. □

3.4.6 Partition a database into components

Algorithm 6 (*Partition*) *Partition* takes as input a database D and returns the list of its components, each labeled with a flag indicating whether it is reflectable. I.e., the output of the algorithm is list of pairs (C, refl), where C is a component of D and *refl* is a Boolean indicating whether C is reflectable or not. We represent this database D using a hashmap, that maps identifiers to their one or two possible values. The algorithm also returns a list of projections on the diagonal to aid Algorithm 7.

For a record ID of database D, let $\mathit{low}(\mathsf{ID})$ and $\mathit{high}(\mathsf{ID})$ be the lower and higher orthogonal projections of point $v = D[\mathsf{ID}]$ on the main diagonal, respectively (see Figure 3.11), i.e., $\mathit{low}(\mathsf{ID})$ and $\mathit{high}(\mathsf{ID})$ are the intersections of the main diagonal with horizontal and vertical lines through point v, where $\mathit{low}(\mathsf{ID}) \preceq \mathit{high}(\mathsf{ID})$. Clearly, a point and its reflection have the same orthogonal projections on the main diagonal.

Algorithm *Partition* (Algorithm 6) is based on the observation that if we project all the points of D onto the main diagonal, the projections of the points of a component are consecutive along the diagonal (see Figure 3.11). The algorithm finds the reflectable components by "walking up" the diagonal and keeping track of the IDs of the points whose projections have been encountered so far. A component is formed once both projections of its points have been seen.

Extra care must be taken in case multiple records are associated with points that have the same projection, say p. In that case, we process first IDs such that p is the higher projection, next IDs such that p is both the upper and lower projection (i.e., $p = D[\mathsf{ID}]$), and finally IDs such that p is the lower projection. See Algorithm 6 for the pseudocode.

Lemma 3.4.4. *Given a database D with R identifiers, each with one or two possible locations, $p, \sigma(p)$, from domain $[N_0] \times [N_1]$, Partition (Algorithm 6) partitions D into its reflectable and nonreflectable components in time $O(\min(R \log R, R + N_0, R + N_1))$.*

Proof. By Lemma 3.3.1 we know that database D has a unique partition. We shall show that

Algorithm 6 *Partition*(D)

```
1: Partition = [] // list of components of database D
2: Projections = [] // list of projections of points on main diagonal
3: Let M be an empty hashmap // maps projections to IDs
4: for all ID ∈ D do
5:     p = D[ID]
6:     Add low(p) and high(p) to Projections
7:     Add ID to M[low(p)]; Add ID to M[high(p)]
8: Component = ∅ // set of IDs of points of current component
9: SeenOnce = ∅ // set of IDs of points of current component for which only one projection has been seen
10: Sort list Projections by ascending order.
11: for all p ∈ Projections do
12:     Order the items ID ∈ M[p] as follows:
13:       first, all ID such that high(ID) = p ≠ low(ID),;
14:       next, all ID such that high(ID) = p = low(ID);
15:       finally, all ID such that high(ID) ≠ p = low(ID);
16:       within each group, order by ID value.
17:     for all ID ∈ M[p] do
18:         if ID ∈ SeenOnce then
19:             Remove ID from SeenOnce
20:             if SeenOnce = ∅ then
21:                 if Component contains a nonreflectable point or a single diagonal point then
22:                     refl = false
23:                 else
24:                     refl = true
25:                 Add (Component, refl) to Partition
26:                 Set Component = ∅
27:         else
28:             Add ID to SeenOnce and to Component
29: Return Partition
```

Algorithm 6 returns the valid partition P of the database. A partition P is invalid if any of these statements holds: (1) P does not contain all points in the database, (2) P contains a component which violates the dominance ordering of the definition or (3) P contains a non-minimal component.

1. P must contain all identifiers of D. Each identifier of D will end up in some component of the partition, when the algorithm eventually processes its projections and adds it to some component.

2. Suppose C_{bad} is a component which contains some point p, such that p dominates some point $q \notin C_{bad}$, but $\sigma(p)$ does not dominate q. Since $q \notin C_{bad}$, that means that

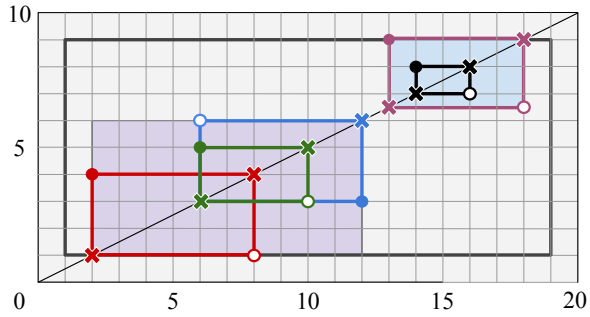


Figure 3.11: Partitioning a database with 5 points (filled circles) over domain $[19] \times [9]$ into a reflectable component (bottom left) and a nonreflectable component (top right) using *Partition* (Algorithm 6). Reflections of points are depicted as empty circles, and projections on the main diagonal as cross marks.

$high(q)$, $low(q) \preceq high(p), low(p)$, as the algorithm traverses the projections in order of dominance and from low to high. However, that implies that $q \preceq \sigma(p)$, which leads to a contradiction. (The proof is similar for any of the four possible violations of the dominance order.)

3. Suppose C_{bad} is a non-minimal component. This means that there exists some subset of points $S \subset C_{bad}$, such that all points $p, \sigma(p)$, where $p \in S$ dominate all points $q, \sigma(q)$ such that $q \in C_{bad} - S$. If that is the case then, for all pairs p, q, where $p \in S, q \in C_{bad} - S$ we have $low(q) \preceq high(q) \preceq low(p) \preceq high(p)$. In this case, Algorithm 6 would have traversed through all the projections (low and high) of points in $C_{bad} - S$ and created a component for them. Thus, C_{bad} would not have been created.

We conclude that Algorithm 6 returns a valid partition of the database identifiers. Regarding run time, generating projections takes $O(R)$ time and sorting them takes either $O(R \log R)$ or $O(\min(R + N_0, R + N_1))$ time depending on whether merge sort or bucket sort is used, respectively. The components are generated traversing up the diagonal and doing $O(1)$ work per projection. Thus, Algorithm 6 takes $O(\min(R \log R, R + N_0, R + N_1))$ time. □

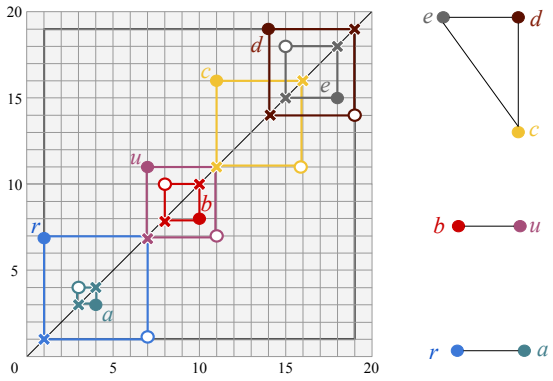


Figure 3.12: (left) Database with 7 records and 3 components over domain $[19] \times [19]$. The true points are shown with filled circles, their reflections with empty circles, and their projections on the main diagonal with cross marks. (right) Graphs for the components constructed by Algorithm 7, where an edge between two records indicates that fixing the point of one record fixes the point of the other record.

3.4.7 Prune the candidate reconstructions

Algorithm 7 utilizes the algorithms discussed so far to get families of possible databases and their (reflectable and non-reflectable) components. To achieve FDR, we must prune the solution set and determine the 1 or 2 possible configurations for each component.

We shall iterate through each component in each family. For each component C, we create a graph G, whose nodes are the identifiers of C. Similarly to *Partition* (Algorithm 6), we find the low and high projections of each identifier on the diagonal. We again "walk-up" the diagonal, adding an edge between ID_1 and ID_2, if the boxes generated by $(low(\mathsf{ID}_1), high(\mathsf{ID}_1))$ and $(low(\mathsf{ID}_2), high(\mathsf{ID}_2))$ intersect in more that one points. More formally, when $high(\mathsf{ID}_1) > low(\mathsf{ID}_2)$ and $high(\mathsf{ID}_2) > low(\mathsf{ID}_1)$. Any identifiers for which $high(\mathsf{ID}) = low(\mathsf{ID})$ are ignored for the purposes of G. The construction of graph G is illustrated in Figure 3.12.

Then, if all identifiers on this component have two possible locations, we pick one identifier and discard one of its locations. Now, we do a depth-first search on graph G starting on an identifier which has only one possible location. On each step of the search we traverse some edge $(\mathsf{ID}_1, \mathsf{ID}_2)$, where at least one of the identifiers, say ID_1 has only one location,

say r. If the other identifier has two u, u', we calculate $\rho_{u,r}$ and $\rho_{u',r}$, and determine which one is consistent with $\mathsf{RM}(D)$. It cannot be that both are consistent, because in that case there wouldn't be an edge between $(\mathsf{ID}_1, \mathsf{ID}_2)$. In case none are consistent, then this database family is invalid.

Algorithm 7 takes as input D's response multiset, $\mathsf{RM}(D)$, and returns a list of families of databases. Each family consists of a database $\hat{D}$ and its partition into components. The attacker produces $\mathsf{E}(D)$ using the output of Algorithm 7. Theorem 3.4.5 shows that that all possible reconstructions of D can be obtained by taking a database $\hat{D}$ returned by the algorithm, applying a rigid motion, and reflecting a subset of its components.

Theorem 3.4.5. *Given the response multiset* ($\mathsf{RM}(D)$) *of some database D with R records over domain $[N_0] \times [N_1]$, Algorithm 7 (*FDR*) returns an encoding of* $\mathsf{E}(D)$*, the set of databases equivalent to D, of size $O(N_0 + N_1)$ in time $O((N_0 + N_1)(RN^2 + R \log R))$, where $N = N_0 N_1$.*

Proof. Line 35 of Algorithm 7 makes sure that we only return databases that are equivalent to D. It remains to show that we return all equivalent databases.

We know from Theorem 3.3.4 that there are at most $O(N_0 + N_1)$ database families in $\mathsf{E}(D)$, each originating from the possible configurations/locations of the extreme points of the database. Each family of databases can be produced using any of its members by reflecting all subsets of reflectable components and then applying the rigid motions of the square.

Thus, we need to show that (1) we find all possible families, (2) we identify all reflectable components of that family, and (3) we produce a database that belongs in each family.

1. By Lemma 3.4.1 we can identify the extreme points and all their possible configurations. Lemmas 3.2.4 and 3.2.5 imply that we can identify at most $4(N_0 + N_1)$ solutions for the relevant extreme points of each configuration. Since there are 2, 3, or 4 extreme points, there is a constant number of possible configurations for them. Thus, we can identify $O(N_0 + N_1)$ database families.

Algorithm 7 FDR(RM(D))

1: Databases = [], RS(D) as the set of RM(D)
2: *PosConfigs* = FindExtremes(RS(D))(Algorithm 3)
3: *Segmentations* = *Segmentation*(*PosConfigs*, RS(D)) (Algorithm 4)
4: Solutions = *Solve*(*PosConfigs*, *Segmentations*, RM(D)) (Algorithm 5)
5: **for** $\hat{D}$ in Solutions **do**
6: Partition = *Partition*($\hat{D}$) (Algorithm 6)
7: **for** each (*Component*, *refl*) in Partition **do**
8: Let Resolved be the set of ID $\in$ *Component* s.t. $|\hat{D}[\mathsf{ID}]| = 1$
9: Let Unresolved be the set of ID $\in$ *Component* s.t. $|\hat{D}[\mathsf{ID}]| = 2$
10: **if** Unresolved $\neq \emptyset$ AND (Resolved $= \emptyset$ OR Resolved only contains points on the diagonal) **then**
11: Pick random ID $\in$ Unresolved and remove one entry of $\hat{D}[\mathsf{ID}]$
12: Remove ID from Unresolved and add it to Resolved
13:
14: Construct graph G with nodes all ID $\in$ *Component*, ignoring points on the diagonal. There is an edge between $\mathsf{ID}_1, \mathsf{ID}_2$ in G, if the boxes defined by ($low(\mathsf{ID}_1)$, $high(\mathsf{ID}_1)$) and ($low(\mathsf{ID}_2)$, $high(\mathsf{ID}_2)$) intersect in more than one point.
15:
16: Let Edges(G) be an iterator of the edges of G given by a depth-first search starting at some $\mathsf{ID}_1 \in$ Resolved.
17: **for** each $(\mathsf{ID}_1, \mathsf{ID}_2) \in$ Edges(G) **do**
18: // At each iteration, one ID is added to Resolved or current solution is discarded
19: **if** $\mathsf{ID}_2 \in$ Unresolved **then**
20: Let $\hat{D}[\mathsf{ID}_1] = r$
21: m = the number of responses in (D) containing ID_1 & ID_2
22: **if** $r \in \hat{D}[\mathsf{ID}_2]$ **then**
23: Remove r from $\hat{D}[\mathsf{ID}_2]$
24: **for** u in $\hat{D}[\mathsf{ID}_2]$ **do**
25: Compute $\rho_{u,r}$ using Equation 3.3
26: **if** $\rho_{u,r} \neq m$ **then**
27: Remove u from $\hat{D}[\mathsf{ID}_2]$
28:
29: **if** $|\hat{D}[\mathsf{ID}_2]| = 1$ **then**
30: Add ID_2 to Resolved and remove it from Unresolved
31: **else**
32: // Here, $\hat{D}[\mathsf{ID}_2] = \emptyset$
33: Go to line 5 (discard the current solution $\hat{D}$)
34: Compute the response multiset RM($\hat{D}$)
35: **if** RM($\hat{D}$) = RM(D) **then**
36: Add $\hat{D}$, Partition to Databases
37: Return Databases

2. Given the solutions for two extreme points, by Lemma 3.4.3 and Lemma 3.2.5 we can identify one or two solutions per identifier. Then, by Lemma 3.4.4 we can identify all reflectable (and non-reflectable) components. Note that the components remain the same even after applying the rigid motions of the square on a database.

3. It remains to show that we can produce a database that belongs to the above family. We know one to two solutions per point and its partition. Each component of the partition has one to two configurations. We shall show that if we fix one (non-diagonal) point in the component, it reduces all other points' possible locations to one.

   Each point uniquely belongs some component C. For each point $p \in C$, there exists some point p' such that the boxes created by $(low(p), high(p))$ and $(low(p'), high(p'))$ intersect. Thus, we can walk up the diagonal on the points' projections, and create a graph G, with nodes the (non-diagonal) points and an edge between two points whose projection boxes intersect.

   We show that if a point r has one possible location, it reduces each of its neighbors u in G to one possible location consistent with the leakage. Let m be the number of different sets in $\mathsf{RS}(D)$ that contain both u and r. We have two cases:

   (a) The boxes intersect : As an example from Figure 3.12, see u and r. WLOG, say that u dominates r, and r anti-dominates $\sigma(u)$. (The proof follows similarly when swapping r and u.) So, $low(r) \preceq low(u) \preceq high(r) \preceq high(u)$.

   We have that $m = r_0(N_0 + 1 - u_0)r_1(N_1 + 1 - u_1)$. We write down the relevant ρ equations, and show that at least one of them is not equal to m, allowing us to trim down the invalid solution.

$$\rho_{u,r} = r_0(N_0 + 1 - u_0)r_1(N_1 + 1 - u_1)$$
$$\rho_{\sigma(u),r} = r_0(N_0 + 1 - u'_0)u'_1(N_1 + 1 - r_1)$$

   Suppose $\rho_{u,r} = \rho_{\sigma(u),r}$, then $u_0 = r_1\frac{N_0+1}{N_1+1}$. Thus, $u'_1 = u_0\frac{N_1+1}{N_0+1} = r_1$. This means that both u and $\sigma(u)$ dominate r. Thus, the boxes could not intersect, and $\rho_{u,r} \neq \rho_{\sigma(u),r}$.

   (b) Point r's box contains the other: As an example from Figure 3.12, u could be

the green point, while r is the blue. WLOG, say that r anti-dominates both u and $\sigma(u)$. (The proof follows similarly in case u dominates r and $\sigma(r)$.) So, $(low(r) \preceq low(u) \preceq high(u) \preceq high(r)$. Again, $m = r_0(N_0 - u_0)u_1(N_1 - r_1)$. We can write down the ρ equations

$$\rho_{u,r} = r_0(N_0 + 1 - u_0)u_1(N_1 + 1 - r_1)$$
$$\rho_{\sigma(u),r} = r_0(N_0 + 1 - u_0')u_1'(N_1 + 1 - r_1)$$

Setting $\rho_{u,r} = \rho_{\sigma(u),r}$, we get that $u_1 = u_0\frac{N_1+1}{N_0+1}$. That means that u is on the diagonal. This is a contradiction as graph G contains no points on the diagonal.

Thus, given one fixed point, we can traverse G and determine the location of every other point. Doing so for all components gives us the family of databases.

First, Algorithm 7 needs to preprocess the leakage, which takes $O(RN^2)$ time. Algorithm 7 runs Algorithms 3, 4 and 5. This takes $O(RN^2 + \min(R^5, RN^2)) = O(RN^2)$. Then, for each database family, we run Algorithm 6 which takes $O(\min(R\log R, R+N_0, R+N_1))$ time. We then generate and traverse G, which takes $O(R\log R)$ time. Note that on each step of the traversal, we need to calculate a query density function. But we could preprocess those in $O(RN^2)$. Note that it suffices to calculate only two query density functions per record. Finally, for each candidate family of databases, we generate a member D' and check if its response set matches $\mathsf{RS}(D)$. It takes $O(RN^2)$ for each family to calculate and check the response set and there are at most $O(N_0 + N_1)$ such families.

Thus, by Lemmas 3.4.1, 3.4.2, 3.4.3, and 3.4.4, we have that Algorithm 7 achieves FDR and runs in time $O(RN^2 + (N_0 + N_1)(\min(R\log R, R + N_0, R + N_1) + R\log R + RN^2))$, which is $O((N_0 + N_1)(RN^2 + R\log R))$.

□

3.5 Experimental Evaluation

Our algorithm leaves a few issues open for empirical exploration: How large a set of seed databases $\mathcal{S}$ would an adversary typically reconstruct and how many components would each of those databases contain? For a rectangular domain, how many components are reflectable? We explore these questions through data representative of what might realistically be stored in an encrypted database with two-dimensional range queries.

Our datasets. We use hospital records from the years 2004, 2008, and 2009 of the Healthcare Cost and Utilization Project's Nationwide Inpatient Sample (HCUP, NIS)[1], seven years, 2012-2018, of Chicago crime locations from the City of Chicago's data portal,[2] and the mobile phone records of Malte Spitz, a German Green party politician.[3]

Prior work also used HCUP data for experimental analysis. The 2009 HCUP data was previously used for the KKNO and LMP attacks, and all three years were used in GLMP19's volume leakage paper [58, 70]. These years were chosen due to their prior use and changes in HCUP's sampling methodology, but other years should give similar results. Also, we explore a new setting for access pattern attacks, geographic datasets indexed by longitude and latitude. We use both Chicago crime data and the phone record locations of Malte Spitz. Each Chicago dataset represents the locations of crimes within a district during a year. The Spitz data were stored by the Deutsche Telekom and contributed by Malte Spitz to Crawdad.

Each year of HCUP data contains a sample of inpatient medical records in the United States. 2004, 2008, and 2009 include data from 1004, 1056, and 1050 hospitals and 8004571, 8158381, and 7810762 records respectively. The NIS is the largest longitudinal hospital care collection in the United States and contains many attributes for each record. We only use a

1. https://www.hcup-us.ahrq.gov/nisoverview.jsp. We did not deanonymize any of the data, our attacks are not designed to deanonymize medical data, and the authors who performed the experiments underwent the HCUP Data Use Agreement training and submitted signed Data Use Agreements.

2. https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2

3. https://crawdad.org/spitz/cellular/20110504/

Attributes	Description	$\mathcal{D}$ 2004	$\mathcal{D}$ 2008	$\mathcal{D}$ 2009
AGE	Age in years	91	91	91
AGEDAY	Age in days	365	365	365
AGE<18	Age $<$ 18	18	18	18
AGE≥18	Age $\geq$ 18	73	73	73
LOS	Length of stay	366	365	365
AMONTH	Admission month	12	12	12
NCH	# chronic conditions	N/A	16	26
NDX	# diagnoses	16	16	26
NPR	# procedures	16	16	26
ZIPINC	Zip code income quartile	4	4	4

Table 3.1: HCUP attributes

small subset of attributes which were used by prior works and come from the Core data file for our analysis. Like KKNO, we divide the age domain into two attributes for minors and adults. While the Agency for Healthcare Research and Quality (AHRQ) provides hospitals with a format and domain for each attribute, many hospitals do not follow the AHRQ guidelines in practice. We use the AHRQ formats for our domain sizes and omit data which do not lie within the domain. We attempt to choose attributes with a variety of data distributions. Also, we avoid pairs of attributes which do not logically make sense to compare (e.g. AGE by AGE_BELOW_18). HCUP attributes are described in Table 3.1.

Chicago was re-districted in 2012, so we only use the 22 districts from years after 2012. The minimum number of crimes in a district across all years was 4162 and the maximum was 22434. The Spitz data contain locations with beginning dates from 166 different days between 8/31/2009 and 2/21/2010. Therefore, we use seven years of Chicago crime data with 22 districts and 7 domains and 166 days of Spitz, leading to a total of 1244 location datasets. The minimum number of phone record locations for a day was 18 and the maximum was 502.

We scale Chicago longitudes to be proportional to the ratio of longitude to latitude in that district to better represent the geometric shapes. For a chosen latitude domain N_0, the minimum longitude domain for a district was around $\frac{N_0}{5}$ and the maximum around $3.6N_0$.

For the HCUP data, we consider databases comprising records from a single hospital and a given a year indexed by two attributes. For the Chicago data, since longitude and latitude are given with up to 12 decimal points, we map a location $(lat,\ long)$ to point (w_0, w_1) of domain $\mathcal{D} = [N_0] \times [N_1]$ by setting $w_0 = \frac{(lat - lat_{min}) \cdot (N_0 - 1)}{lat_{max} - lat_{min}} + 1$ and $w_1 = \frac{(long - long_{min}) \cdot (N_1 - 1)}{long_{max} - long_{min}} + 1$, where division is rounded to the nearest integer. We set $N_0 = 9, 19, 39, 59, 99, 199,$ and 1999, and set N_1 so that the domain preserves the ratio of longitude range to latitude range of the district. The resulting domains are square for only 6 districts. For Spitz data we choose to use square geographic domains. We use the actual longitudes and latitudes multiplied by 100 as integers and center the smaller range in the square domain. The maximum $N_0 = N_1$ we observe is 677.

To generate the leakage for our attack, for each database, we build the response multiset by querying each possible range $(c, d) \in \mathcal{D}^2$ such that $c \preceq d$.

Dataset and attributes		Domain	# DBs by # of reflectable components	# DBs
NIS 2004	AGE & LOS	91x366	1004/0...	1004
	AGEDAY & ZIPINC	365x4	677/0...	677
	AGE<18 & NPR	18x16	972/0...	972
	AMONTH & ZIPINC	12x4	948/0...	948
	NDX & NPR	16x16	0/997/7/0...	1004
NIS 2008	AGE≥18 & NPR	73x18	1055/0...	1055
	AMONTH & NCH	12x16	1005/0...	1005
	NCH & NDX	16x16	0/1054/1/0/1/0...	1056
	NCH & NPR	16x16	0/1053/3/0...	1056
	NDX & NPR	16x16	0/1052/4/0...	1056
NIS 2009	AGE<18 & LOS	18x366	968/0...	968
	AMONTH & AGEDAY	12x365	644/0...	644
	NCH & NDX	26x26	0/1043/7/0...	1050
	NCH & NPR	26x26	0/1049/1/0...	1050
	NDX & NPR	26x26	0/1017/3/0...	1050
Chicago	LAT & LONG	9, 19, 39, 59, 99, 199, 1999	1072/6/0...	1078
Spitz	LAT & LONG	$\leq 677 \times 677$	0/117/35/9/3/0 /0/0/2/0...	166

Table 3.2: Results of our experiments on real-world datasets. In the third column, $n_0/n_1/n_2/\ldots$ means that n_i databases have i reflectable components, $i = 0, 1, 2, \ldots$.

Our findings. Our results are shown in Table 3.2. Recall that the reconstruction returns a set $\mathcal{S}$ of seed databases that generates a family of $\sum_{D \in \mathcal{S}} 4 \cdot 2^{r(D)}$ equivalent databases, where $r(D)$ denotes the number of reflectable components of D. For all databases, our reconstruction found a single seed database (i.e., $|\mathcal{S}| = 1$). Thus, we report the number of reflectable components for this database. The number of reflectable components for the Chicago data is consistent for all chosen domains, so we compress our seven longitude domains into a single row in the table. The majority of our datasets leaked an equivalent family of minimal size (15713 out of 15792 instances). Across all our experimental attributes, a total of 61 datasets leaked a family of databases of size 16, 9 leaked a family of size 32, 7 leaked a family of size 64, and 2 leaked a family of size 1024.

Whether a database has a rectangular or square domain is a major determining factor in the number of equivalent databases. In our experiments, all components in rectangular databases could be fixed to not be reflected, suggesting that few real rectangular databases with similar attributes would have a large number of equivalent databases. This accounts for 8345 of our datasets. In the square case, the number of reflectable components varies with the distribution of the data. Among 7322 square HCUP datasets, there were 26 datasets with two reflectable components and only one with 4 components. However, the square Spitz datasets were frequently distributed along the main diagonal, leading to larger families of equivalent databases. Of the 166 Spitz datasets, around 1/4 of them had ≥ 2 reflectable components, with a maximum of eight components. To illustrate the structure of the Spitz data, we show in Figure 3.1 the phone record locations for 08/31/2009 and our reconstruction[4]. Our algorithm finds 8 reflectable components, (4 single points and 4 multi-point components shown as shaded squares), resulting in $4 \cdot 2^8 = 1024$ equivalent databases.

Conclusions. The data show that one may rarely see symmetries arising from multiple seed databases or from reflectable components in rectangular domains. These were expected, as

4. Figure 3.1 map source: Google Maps

those symmetries correspond to number-theoretic coincidences in the data. We also conclude that multiple components will plausibly appear in real data. Some data types tend to have a single component, while other types have some larger number, and sporadic examples with several components can occur. For example, when the Spitz data was divided into days, multiple components could arise when the travel was roughly diagonal. Over a longer period, however, the diagonal structure was lost. We ran an additional test using a database of Spitz records from every day and found only a single reflectable component. In the Chicago crime data, the distribution of events was also rarely so well-structured. In the HCUP datasets, we observed that occasional datasets with correlated attributes could have multiple components, but most instances lacked this type of diagonal distribution.

3.6 Automatically Finding DB in E(DB)

Our attack in Section 3.4, and those of prior work [58, 70] only recover $\mathsf{E}(D)$ and not D. Indeed, this is the best one can hope for when giving a worse-case algorithm. However in practice it is intuitive that an attacker could sometimes do better by observing the distribution of the data recovered and applying one of the allowed symmetries to best match the expected distribution. This section formalizes such an attack for one and two-dimensional cases.

Attack setting. We assume that an attacker has recovered $\mathsf{E}(D_{\text{test}})$ and aims to determine which member of that set is the correct database D_{test}. With no context this is impossible, so we assume that the adversary has auxiliary knowledge of the data through a similar dataset D_{train}. In our experiments below, we give the attack auxiliary data in the form of a histogram of D_{train}, the mean of D_{train}, or a single point from D_{test}.

This attack setting is not totally realistic because if an attack had such auxiliary knowledge then it would probably also apply it during the initial phase that recovered $\mathsf{E}(D_{\text{test}})$, but doing so is an open problem that requires different ideas. For now we interpret our experiments here as determining if sometimes recovering $\mathsf{E}(D_{\text{test}})$ essentially allows recovering D_{test} itself.

Our attack. We will first consider the case where the attack can view a histogram $\mathsf{H}_{\text{train}}$ of D_{train}. In one dimension, $\mathsf{H}_{\text{train}}$ is on the domain $\mathcal{D}$ and in two dimensions, $\mathsf{H}_{\text{train}}$ is a joint histogram over $\mathcal{D}$. Our attack recovers the family of equivalent databases (either through the KKNO attack in one dimension or the attack from Section 3.4 in two dimensions) and tries to identity which database in the family is the true D_{test}. Given $\mathsf{E}(D_{\text{test}}) = \{D_0, \ldots, D_k\}$, the attacker will compute k histograms $\mathsf{H}_0, \ldots, \mathsf{H}_k$ and select the D_i with the H_i that minimizes the ***mean squared error*** with respect to $\mathsf{H}_{\text{train}}$. More formally, define

$$\mathsf{MSE}(\mathsf{H}_{\text{train}}, \mathsf{H}_i) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} (\mathsf{H}_{\text{train}}(x) - \mathsf{H}_i(x))^2.$$

The attack selects D_i corresponding to H_i with the minimum $\mathsf{MSE}(\mathsf{H}_{\text{train}}, \mathsf{H}_i)$.

Next, we also consider the case of a weaker adversary, who knows only the mean from the similar database, μ_{train}. In that case, the attack selects the database with the closest mean to the training data, minimizing $|\mu_{\text{train}} - \mu_i|$. Because the means and histograms of geographic data in terms of latitude and longitude seem less realistic to be available publicly than the means and histograms of medical attributes, we consider another weak adversary who only knows a single location, p, in the test database. The attack outputs a guess uniformly at random from the equivalent databases which contain p to identify D_{test} in $\mathsf{E}(D_{\text{test}})$.

To experimentally test the MSE approach, we take the overall data distribution across all hospitals for a single attribute in one dimension or a pair of attributes in two dimensions of 2008 HCUP data as our training distribution D_{train} and use each hospital from the 2009 HCUP data for those attributes as a D_{test}. We use HCUP 2008 domain sizes and exclude HCUP 2009 data which exceed those domains. For an adversary with only the mean, we take the publicly available mean from 2008 for each attribute, or we calculate the mean across all hospitals in 2008 for attributes where the mean is not reported online. We evaluate an adversary who knows only a single point in the database with Chicago Crime and Spitz data, and we choose the known location uniformly from the locations in each database.

Attribute	$\mathcal{D}$	Acc. MSE	Acc. μ	# DBs
LOS	366	1.00	1.00	1049
AGEDAY	365	0.91	0.92	645
AGE	125	0.99	0.73	1049
AGE_18_OR_GREATER	107	0.96	0.90	1049
AGE_BELOW_18	18	0.75	0.74	1049
NCH	16	1.00	1.00	1050
NDX	16	0.83	0.68	1050
NPR	16	1.00	1.00	1050
AMONTH	12	0.66	0.66	1000
ZIPINC_QRTL	4	0.72	0.74	1049

Table 3.3: Symmetry breaking for 1D range queries.

We run experiments on both 1D and 2D databases. We present our results in one dimension on just HCUP data in Table 3.3. We can see that for both the MSE and mean attacks, the reflection is typically easy to remove. While the attack with a full histogram outperforms the attack with only the mean, they are both effective. For smaller domains these approaches do not work as well and for other domains like admission month (AMONTH), where the data are fairly uniform, it is harder to accurately determine the symmetry.

We present our two-dimensional results in Table 3.4. We note that the joint accuracy has a baseline of $1/8 = 0.125$ when there is one reflectable component. For our HCUP databases the attacks did much better than the baseline, but were not completely accurate. For the location data with a single known point, it was possible to find D in $\mathsf{E}(D)$ with high probability for Chicago data with large domains. Denser databases, like the Chicago data with small domains, and databases with many reflectable components, like the Spitz dataset, still were significantly better than the baseline but had worse performance.

Attributes	$\mathcal{D}$	Acc. MSE	Acc. μ	Acc. p	# DBs
NCH & NDX	16×16	0.743	0.683	N/A	1050
NCH & NPR	16×16	0.935	0.927	N/A	1050
NDX & NPR	16×16	0.668	0.580	N/A	1050
Chi LAT & LONG	9	N/A	N/A	0.364	154
Chi LAT & LONG	19	N/A	N/A	0.467	154
Chi LAT & LONG	39	N/A	N/A	0.506	154
Chi LAT & LONG	59	N/A	N/A	0.571	154
Chi LAT & LONG	99	N/A	N/A	0.721	154
Chi LAT & LONG	199	N/A	N/A	0.890	154
Chi LAT & LONG	1999	N/A	N/A	1.0	154
Spitz LAT & LONG	≤ 677	N/A	N/A	0.524	166

Table 3.4: Symmetry breaking for 2D range queries.

3.7 Conclusion and Future Work

We have shown that full database reconstruction from responses to range queries is much more complex in two dimensions than one. Indeed, going from 1D to 2D, the worst-case number of databases that produce equivalent leakage jumps from constant to exponential (in the database size). Despite this limitation, we develop a poly-time reconstruction algorithm that computes and encodes all databases with equivalent leakage in poly-space. We implement our attack and demonstrate that the configurations that lead to a large number of equivalent databases are present in real data. As new approaches to search on encrypted databases are being proposed, our work identifies specific technical challenges to address in the development of schemes for range search resilient to attacks.

CHAPTER 4

RECONSTRUCTING WITH LESS: LEAKAGE ABUSE ATTACKS IN TWO DIMENSIONS

This chapter first appeared as conference publication by the same title with minor modifications in CCS 2021 [87]. It is joint work with Evangelia Anna Markatou, Roberto Tamassia, and William Schor.

4.1 Introduction

The growing adoption of cloud computing and storage in the past two decades has been accompanied by a corresponding increase in data breaches. Encrypted cloud storage reduces the risk of such breaches. In particular, searchable encryption provides a practical solution for processing queries over encrypted data without the need for decryption.

For the sake of efficiency, searchable encryption schemes sacrifice full security by leaking some information about the queries and their responses.

As a consequence, the underlying data is vulnerable to inference attacks from this leakage. To defend against such attacks, a variety of mitigation techniques have been developed. Works on encryption schemes and mitigation techniques include [22,37,42,43,57–60,64,70,72–75,88,89,100,102,104]. Systems implementing these types of schemes have been developed in both academia (e.g., [103,106]) and in industry (e.g., [101]). Regarding reconstruction attacks, see, e.g., [22,43,58–60,64,70,72–75,88,100,104].

The following standard types of leakage occur in searchable encryption schemes. A scheme leaks the ***access pattern*** if the adversary observes the encrypted records returned in response to queries. A scheme leaks the ***search pattern*** if the adversary can distinguish if a query has been previously issued, i.e., can assign a unique query identifier to each distinct query.

This work considers an encrypted database with two attributes, referred to as a ***two-***

dimensional (2D) database to which ***range queries*** are issued. We assume a passive persistent adversary who observes the entire access pattern leakage, i.e., all possible responses of queries, and a subset of the search pattern leakage. Our adversary aims to reconstruct the order of the database records in the two dimensions (attributes) using solely the access pattern, a problem called ***order reconstruction (OR)***. The adversary then performs an approximate reconstruction of the (attribute) values of the database records by using the partial search pattern observed, a problem called ***approximate database reconstruction (ADR)***. A more ambitious goal is ***full database reconstruction (FDR)***, which aims at computing the exact record values, up to unavoidable symmetries and other information theoretic limitations.

4.1.1 Contributions

Previous work on reconstruction attacks from range queries on 2D databases [43] assumes that the adversary has knowledge of the entire access and search pattern leakage (i.e., has seen all possible queries and their responses) and uses both forms of leakage to perform an attack that reconstructs the record values in polynomial time, up to inherent information theoretic limitations.

In contrast to [43], we investigate what information is recoverable from 2D range queries when given only a fraction of the possible responses. We make progress in this direction with the following contributions:

1. We show that order reconstruction faces additional information theoretic limitations when given only access pattern leakage. We describe and prove a ***complete characterization*** of the family of databases that have the same access pattern leakage.

2. We present an ***order reconstruction attack*** that allows an adversary with the entire access pattern to build a linear-space representation of the family of databases in poly-time.

3. We design a ***distribution-agnostic approximate database reconstruction attack*** that reconstructs record values given the order of the records, and the search and access pattern leakage from any number of queries drawn from an unknown distribution.

4. We ***empirically evaluate*** the effectiveness of our attacks on real-world datasets under different query distributions.

5. We develop new ***combinatorial and geometric concepts and algorithms*** related to point reconstruction from range queries that may be of independent interest.

Our work provides the ***first order reconstruction attack in 2D from access pattern leakage*** and the ***first approximate reconstruction attack in 2D from partial search pattern leakage and an unknown query distribution***. Our order reconstruction attack does not require knowledge of the domain size and, instead, gives us a lower bound of the domain size. This attack is also a full database reconstruction attack for the case when the 1D horizontal and vertical projections of the points are ***dense***, i.e., have a record for every domain value.

Our work improves over the full database reconstruction attack of [43], where the adversary observes both access pattern and search pattern from all possible queries on the database. This previous attack fails when even a single query is missing. In contrast, we demonstrate that an adversary can still infer much about the original data with significantly less information.

Our approximate database reconstruction (ADR) attack can be viewed as the 2D analogue of the work on attacks on 1D databases reported in [73]. To apply previous approximation approaches that assume knowledge of the order to 2D databases, we must completely characterize order reconstruction in 2D. However, much like FDR does not trivially extend from the 1D to 2D setting, our order reconstruction method demonstrates an exponential increase in the number of indistinguishable point configurations in the 2D setting. Thus, we cannot simply generalize 1D techniques to 2D. We re-examine a number of support-size

estimators to better suit our problem. We emphasize that while our techniques are distribution agnostic (i.e., they do not require knowledge of the query distribution), certain distributions prevent the observation of a large fraction of responses and records (i.e., a distribution where only a few queries have nonzero probability) and thus place severe information theoretic limits on the accuracy of any approximate reconstruction method. In Section 4.6 we examine different non-parametric estimators and their efficacy under different query distributions. In Section 4.7 we build a complex nonlinear system of equations to model the problem instead of the linear system of [73].

4.1.2 EDBs and 2D Range Queries

There are a number of schemes that support two-dimensional range queries over encrypted data. All existing schemes leak access and search pattern, and many leak strictly more information. Our work is motivated by the need to understand what can be learned from information leakage that seems unavoidable without employing the use of oblivious RAMs (ORAMs) [55] or fully homomorphic encryption [50], both of which incur significant overhead.

Shi et al. [114] designed a scheme called Multidimensional Range Query over Encrypted Data (MRQED) that leverages public key encryption. Although their model is different, their scheme leaks strictly more than access and search pattern. MRQED achieves "match-revealing" security which reveals the attributes of the range query when the query is successfully decrypted. The scheme builds a binary tree on the values of each dimension, and releases public keys corresponding to the nodes that "cover" the range of interest. The server learns both search and access of the query, the plaintexts of the matching records, and structural information about range query issued. Maple is a tree-based public-key multi-dimensional range searchable encryption scheme [125]. Its goal is to provide single-dimensional privacy which mitigates one-dimensional database reconstruction attacks. In addition to leaking access and search pattern, they also leak the nodes accessed when traversing the range tree

and the values of each queried range. Recently, Kamara et al. gave constructions for schemes that support conjunctive SQL queries with a reduced leakage profile [25, 65, 132].

One may also consider an index-based construction described in [43] that is built on top of a multi-keyword searchable symmetric encryption (SSE) scheme, like Cash et al. [24].

To mitigate 1D attacks and avoid leaking information about individual columns, one can precompute a joint index of all possible 2D queries and encrypt the resulting index. When a 2D query is issued, only records matching both dimensions will be returned and the leakage is precisely the leakage of the underlying SSE scheme used.

4.1.3 Comparison with Prior Work

In the following, we denote with N the size of the domain of the database points. We present the first order reconstruction and the first approximate database reconstruction in 2D; our attacks only require a strict subset of the leakage used by the 2D attacks in [43]. Previous 2D attacks require the multiset of access patterns, which can be obtained with $O(N^4 \log N)$ uniformly random queries. In contrast, our order reconstruction attack takes as input the set of access pattern leakage, which can be obtained with $O(N^2 \log N)$ uniformly random queries. Our approximate database reconstruction attack requires search and access pattern leakage, however, we are able to recover information with small relative error with as few as 4% of the possible queries. Table 4.1 compares our results with previous work, where Dense1D denotes a 2D database whose horizontal and vertical projections are each a dense 1D database.

Table 4.1 compares our attacks with selected related work.

4.2 Preliminaries

Input to Attacks. The ***query pattern*** or ***search pattern*** of a query $q = (c, d)$ is defined to be a query-specific token $\mathsf{QP}(D, q) = t$, where $t \in \left[\binom{N_0+1}{2}\binom{N_1+1}{2}\right]$. We assume a 1-1 correspondence between queries and tokens.

	Queries		Assumptions		Leakage	Attack		
	1D range	2D range	Query distrib.	Database	Search pattern	**OR**	FDR	**ADR**
Kellaris+ [70]	✓		Uniform	Any		✓	✓	✓
Lacharité+ [75]	✓		Unknown	Dense		✓	✓	
Grubbs+ [59]	✓		Uniform	Any		✓	✓	✓
Markatou+ [88]	✓		Unknown	Any		✓		
Markatou+ [88]	✓		Unknown	Any	✓		✓	
Kornaropoulos+ [73]	✓		Unknown	Any	✓			✓
Falzon+ [43]		✓	Unknown	Any	✓		✓	
Falzon+ [43]		✓	Known	Any			✓	
This Work		✓	**Unknown**	**Any**		✓		
This Work		✓	**Unknown**	**Any**	✓			✓
This Work		✓	**Unknown**	**Dense1D**		✓	✓	

Table 4.1: Comparison of our attack with related ones that assume access pattern leakage.

Our order reconstruction algorithm (Sections 4.4-4.5) takes $\mathsf{RS}(D)$ as input. Our approximate database reconstruction algorithm (Sections 4.6–4.7) takes as input a submultiset of $\mathsf{RM}(D)$ (i.e., the query and access pattern) observed by the adversary for any number of queries drawn from an arbitrary distribution.

Threat model. We study the security of encrypted database schemes that support two-dimensional range queries and which leak the access pattern and query pattern of each query. We consider a ***passive***, ***honest-but-curious***, ***persistent*** adversary that has compromised the database management system or the client-server communication channel, and can observe the leakage over an extended period of time. Our order reconstruction attack considers an adversary that takes $\mathsf{RS}(D)$ as input and wishes to compute the order of all records. Our second attack considers an adversary that knows the search tokens and responses of some

sample of queries, as well as the order of all the records, and wishes to approximate the domain value of each record.

Assumptions and reconstruction attacks. We explore reconstruction under a few different assumptions. In Section 4.5 we assume the adversary knows the full response set $\mathsf{RS}(D)$. In Section 4.7 we assume the adversary knows the domain, but we make no assumption about the number of queries that it may have observed or the distribution from which queries are drawn; the adversary has no knowledge of the distribution.

In Section 4.7, we give a method for estimating the values of the database given only query pattern. In particular, given the order of points in D and a multiset of search token and response pairs (where each pair corresponds to one of the observed queries) we preform approximate database reconstruction (ADR).

4.2.1 Query Densities

We use the generalized notion of query densities of points and point sets in two-dimensions presented in Chapter 3 and [43], which extends the methods in [70] for computing the number of unique queries whose responses contain a given set of points. By observing sufficiently many query responses of uniformly random queries, one can recover the value of a point x by computing the proportion of responses that the identifier of x appears in.

Definition 12 (Chapter 3,[43])**.** *Let $\mathcal{D} = [N_0] \times [N_1]$. The **query density** of a point $x \in \mathcal{D}$ is defined as*

$$\rho_x = \left| \{(c, d) \in \mathcal{D}^2 : c \preceq x \preceq d\} \right|.$$

The query density a set of points $S \subseteq \mathcal{D}$ defined as

$$\rho_S = \left| \{(c, d) \in \mathcal{D}^2 : \forall x \in S,\ c \preceq x \preceq d\} \right|.$$

Thus, these are the number of queries that contain x or all points in S, respectively.

Given a point $x = (x_0, x_1) \in \mathcal{D}$, the formula for computing ρ_x is

$$\rho_x = x_0 \cdot x_1 \cdot (N_0 + 1 - x_0) \cdot (N_1 + 1 - x_1). \tag{4.1}$$

The query density ρ_S of a set of points $S \subseteq \mathcal{D}$ is

$$\rho_S = (\min_{x \in S} x_0)(\min_{y \in S} y_1)(N_0 + 1 - \max_{z \in S} z_0)(N_1 + 1 - \max_{w \in S} w_1). \tag{4.2}$$

4.3 Order and Equivalent Databases

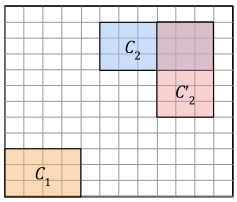
Before developing our attacks, we present our results on the information-theoretic limitations of order reconstruction.

As shown in Chapter 3 and [43], given some database D we can generate a database D' that is equivalent with respect to the response multiset by rotating/reflecting D according to the symmetries of the square and by independently flipping the reflectable components across the main diagonal.

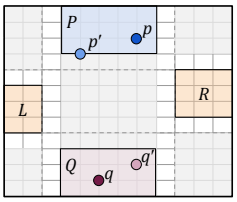
We recall Lemma 3.3.2 and note that if D and D' are equivalent with respect to the response multiset, then they are equivalent with respect to the response set. The converse is not necessarily true. We show in Propositions 4.3.1 and 4.3.2 (Figure 4.1) that there are two additional symmetries that produce equivalent databases with respect to the response set.

Definition 13. *A pair of points (p, q) of a database D is an **antipodal pair** if for every point $r \in D - \{p, q\}$ we have (1) $q_1 < r_1 < p_1$ and (2) either $r_0 < \min(p_0, q_0)$ or $r_0 > \max(p_0, q_0)$. See Figure 4.1b.*

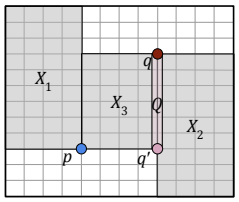
Definition 14. *A pair (p, q) of points of a database D are said to be a **close pair** if q minimally dominates p, and there exists no point $r \in D - \{p, q\}$ such that r anti-dominates p or r is anti-dominated by q or r is between p and q. See Figure 4.1c.*



(a) Illustration of Definition 11 and Proposition 3.3.2. C_1 and C_2 are components of D. Flipping C_2 along the diagonal yields an equivalent database with respect to the response multi-set.



(b) Illustration of Definition 13 and Proposition 4.3.1. Points p and q are an antipodal pair. Each remaining point is in L or R. Replacing p with $p' \in P$ and q with $q' \in Q$ gives an equivalent database with respect to the response set.



(c) Illustration of Definition 14 and Proposition 4.3.2. Points p and q are a close pair. There are no points in regions X_1, X_2 or X_3. Replacing q with any $q' \in Q$ yields an equivalent database with respect to the response set.

Figure 4.1: Examples of transformations that yield equivalent databases with respect to the response set (Definition 6).

The following proposition, illustrated in Figure 4.1b, shows that one cannot infer the horizontal ordering of an antipodal pair from the response set.

Proposition 4.3.1. *Let D be a database from domain $\mathcal{D}$ that contains an antipodal pair (p, q). Let V be the widest vertical strip of points of $\mathcal{D}$ that contains p and q, and let P and Q be the tallest horizontal strips of V containing p and q, respectively, but no other point of D. Let D' be the database obtained from D by replacing p with another point, p', of P and q with another point, q', of Q. We have that databases D and D' are equivalent with respect to the response set, i.e., $\mathsf{RS}(D) = \mathsf{RS}(D')$.*

Proof. Let $D[i] = p$, $D[j] = q$, $D'[i] = p'$, and $D'[j] = q'$. We first show that $\mathsf{RS}(D) \subseteq \mathsf{RS}(D')$. Consider a response A in $\mathsf{RS}(D)$ that contains i and not j. We will exemplify a query to D' with response A. Consider the set $B = (A - \{i\})$. Since $D[i]$ has a unique maximal value in the second coordinate the set B must be an element of $\mathsf{RS}(D)$. By assumption, $\mathsf{RS}(D - \{p, q\}) = \mathsf{RS}(D' - \{p', q'\})$ and so we have that $B \in \mathsf{RS}(D')$. Let $(c, d) \in \mathcal{D}^2$ be a query that generates the response B in D'. Now consider the query $((min_0, 1), (max_0, d_1))$ where $min_0 = \min(c_0, p_0, p'_0)$ and $max_0 = \max(d_0, p_0, p'_0)$. Since the only additional identifier contained in this region is i, then the response generated by this query is $A = B \cup \{i\}$ which

implies $A \in \mathsf{RS}(D')$. A similar argument holds for queries that contain j and not i, as well as queries that contain both i and j, which concludes the forward direction of the proof. One can also extend this reasoning to show that $\mathsf{RS}(D') \subseteq \mathsf{RS}(D)$. □

By Proposition 4.3.1, the two points of the antipodal pair (p, q) of D and of the corresponding antipodal pair (p', q') of D' can be ordered, reverse ordered, or collinear in the horizontal dimension and these three orderings cannot be distinguished using $\mathsf{RS}(D)$.

Proposition 4.3.2. *Let D be a database from domain $\mathcal{D}$ that has a close pair (p, q). Let D' be the database obtained from D by replacing q with any point q' such that $q'_0 = q_0$ and $p_1 \leq q'_1 \leq q_1$. Then D and D' are equivalent with respect to the response set, i.e., $\mathsf{RS}(D) = \mathsf{RS}(D')$.*

Proof. Let $D[i] = q$ and $D'[i] = q'$. By assumption $\mathsf{RS}(D - \{q\}) = \mathsf{RS}(D' - \{q'\})$. We first show that $\mathsf{RS}(D) \subseteq \mathsf{RS}(D')$. We claim that for any response $A \cup \{i\}$ in $\mathsf{RS}(D)$ there exists a response $A \cup \{i\} \in \mathsf{RS}(D')$. Let $A \cup \{i\}$ be a response in $\mathsf{RS}(D)$ and let $(c, d) \in \mathcal{D}^2$ be a query to D that produces such a response. We will consider two possible cases and in each case explicitly give a query to D' that must result in the response $A \cup \{i\}$.

Case 1: $p_0 < c_0$. Consider the query $((c_0, min_1), d)$ issued to D' such that $min_1 = \min(q'_1, c_1)$. If $min_1 = c_1$ then

$$\mathsf{Resp}(D', ((c_0, min_1), d)) = \mathsf{Resp}(D', (c, d)) = A \cup \{i\}$$

since all points $r \in A$ are identical in both D and D' and q' is contained in this query. Else if $min_1 = q'_1$ then by definition of close pair, q, q' must minimally dominate p. So no additional points beside q' are contained in the response generated by $((c_0, min_1), d)$ thus $\mathsf{Resp}(D', ((c_0, min_1), d)) = A \cup \{i\}$.

Case 2: $c_0 \leq p_0$. Since the query (c, d) contains q then we have $c \preceq p$ and $q \preceq d$. Moreover $p \preceq q' \preceq q$ and so $\mathsf{Resp}(D', (c, d)) = A \cup \{i\}$.

That proves the forward direction of the proof. A similar argument holds for the backward direction and we conclude that $\mathsf{RS}(D) = \mathsf{RS}(D')$. □

Combining Propositions 3.3.2, 4.3.1 and 4.3.2, we characterize the information-theoretic limitations of order reconstruction.

Theorem 4.3.3. *Let D be a two-dimensional database. The set of point orderings $\mathsf{E_o}(D)$ can be obtained from the dominance graph G, the anti-dominance graph G', the antipodal pair (if it exists), and the set of close pairs of D by means of the following transformations:*

1. *Flipping the direction of G and/or a subset of components of G' according to Proposition 3.3.2.*

2. *If D contains an antipodal pair, add or remove one or two edges from G or G' to make the pair collinear or switch their relationship from strict dominance to strict anti-dominance or vice versa.*

3. *For each close pair in D, add or remove one or two edges from G or G' to make them collinear or put them in a strict dominance relationship.*

We prove Theorem 4.3.3 in Section 4.4.1. The equivalent configurations of Propositions 4.3.1 and 4.3.2 arise only with respect to the response set. The multiplicity information from the response multiset provided by the search pattern resolves them. Indeed, Theorem 4.3.3 adds transformations (2) and (3) to transformation (1) given in [43].

4.3.1 Chains and Antichains

Our order reconstruction algorithm uses the concepts of chains and antichains of the dominance and anti-dominance relations for points in the plane [46, 123].

A set of points $S \subseteq \mathcal{D}$ is a ***chain*** if any two points $x, w \in S$ are in a dominance relationship i.e. $x \preceq w$ or $w \preceq x$. A subset of points $A \subseteq \mathcal{D}$ is an ***antichain*** if for any two points $x, w \in A$ neither $x \preceq w$ nor $w \preceq x$. Let $D \subseteq \mathcal{D}$ be a set of points. The ***height*** of a point $x \in D$ is the length of the longest chain in D with x as the maximal element. Note that two points

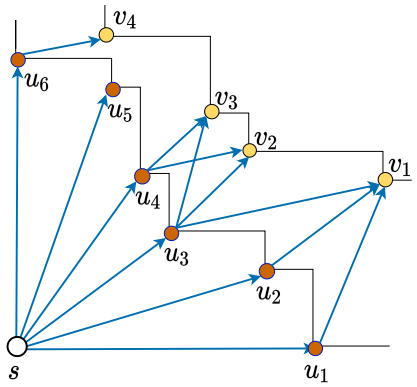


Figure 4.2: Example of a dominance graph and its associated canonical antichain partition comprising antichains $A_0 = \{s\}$, $A_1 = \{u_1, \cdots u_6\}$, and $A_2 = \{v_1, \cdots v_4\}$.

of the same height cannot have a dominance relation. Thus, the set of all points in D with the same height yields a partition $\mathcal{A}$ of D into antichains, namely the ***canonical antichain partition***. We denote the canonical antichain partition by $(A_0, A_1, \ldots, A_L)$ where A_i is the set of points at height i.

Let D be a database and let (G, G') be the dominance and anti-dominance graphs of D. Now note that the paths in the dominance graph correspond to chains in D. Formally, if $(u_1, u_2, \ldots, u_\ell)$ is a path of record IDs in G, then $D[u_1] \preceq D[u_2] \preceq \cdots \preceq D[u_\ell]$ and $\{D[u_1], D[u_2], \ldots, D[u_\ell]\}$ forms a chain in D. By definition the edges of G represent the minimal dominance relations of the points in D and thus determining the length of a longest possible path in G from a source to $u \in [R]$ is equivalent to determining the height of point $D[u]$ in the database. This gives us a nice way of partitioning the IDs such that the partition corresponds to the canonical antichain partition. Formally, if s is a source of G then $D[s]$ has height 0. And if $S_i \subseteq [R]$ is the set of IDs in G that have a maximum distance of i from any sink, then the canonical antichain partition of D is given by $A_i = \{D[a] \ : \ a \in S_i\}$.

For an example, see Figure 4.2. Since G is acyclic we can compute these longest paths efficiently. For convenience we may use A_i to instead refer to the IDs of points within each partition of the canonical antichain.

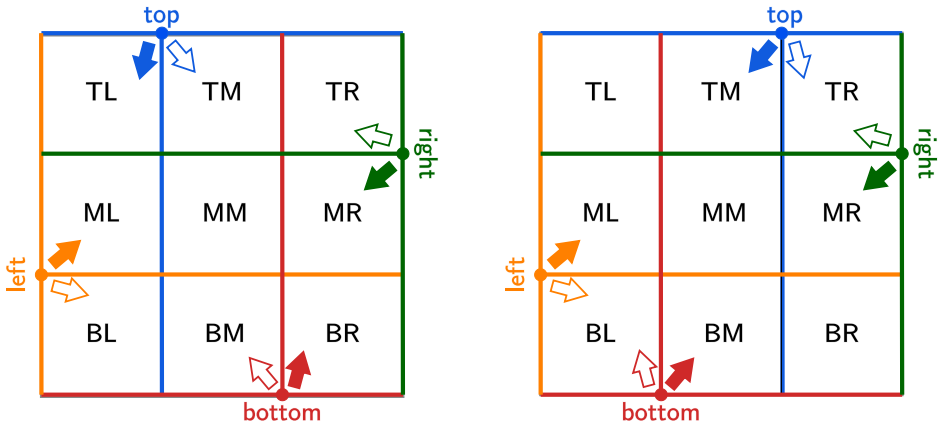
These observations are crucial in the design of our OR algorithm. E.g., we construct the

dominance graph starting at the IDs of points with height 0. We then compute the partition on IDs that correspond to the canonical antichain partition and use this partition to construct the anti-dominance graph.

4.4 Overview of Order Reconstruction

A high-level intuitive explanation for our order reconstruction algorithm is schematically illustrated in Figure 4.3, where we show a database that has distinct extreme points $\mathsf{left}, \mathsf{right}, \mathsf{top}$ and bottom. We assume, without loss of generality, that $\mathsf{left} \preceq \mathsf{right}$. The two parts of the figure distinguish the cases where top is to the left or right of bottom, respectively. By symmetry, these two cases cover all the possible configurations of the extreme points. For simplicity, we assume that none of the remaining points are horizontally or vertically aligned with each other or the extreme points. Thus, only the four extreme points are on the boundary of the rectangle occupied by the database points. The OR algorithm presented in the next section will remove these simplifying assumptions and reconstruct an arbitrary database. A first building block of our OR algorithm finds such extreme points from the response set. We leverage an algorithm from [43] to find these extreme points, however our techniques diverge considerably from [43] after this. Whereas they solve a system of degree four polynomials with full knowledge of $\mathsf{RM}(D)$, our OR algorithm determines the relationships between pairs of records using only set containment observed in $\mathsf{RS}(D)$.

Partition of the Database into Regions. By drawing horizontal and vertical lines through the extreme points, we partition the database points into nine regions labeled XY for $\mathsf{X} \in \{\mathsf{T}, \mathsf{M}, \mathsf{B}\}$ and $\mathsf{Y} \in \{\mathsf{L}, \mathsf{M}, \mathsf{R}\}$, where T, B, L, R, and M stand for top, bottom, left, right, and middle, respectively. Note that some of these regions may be empty. We can compute the points in each region from the response set by finding minimal responses that contain certain pairs and triplets of extreme points and performing intersections and differences of such responses with each other and the entire database. We show how to compute the rows



(a) top to the left of bottom (b) top to the right of bottom

Figure 4.3: Partition of the database points into nine regions induced by the extreme points.

and columns, from which a region can be computed by intersecting its row with its column. The middle row and column are the minimal response containing left and bottom and the minimal response containing top and bottom, respectively. The other rows and columns are obtained by computing the minimal response containing the triplet of extreme points opposite to the column and subtracting this response from the database. For example, the left column is obtained by subtracting from the database the minimal response containing top, right, and bottom.

(Anti-)Dominance with a Corner. Consider a subset S of the database containing a dominance corner, s, defined as a point that dominates or is dominated by all other points of S. For example, point left is a dominance corner for the points in region ML in Figure 4.3a. Another building block of our algorithm is a method that given S and s, computes all pairs of points of S that have a dominance relation. By symmetry, the same methods compute the anti-dominance relation pairs for a subset of points that admits a similarly defined anti-dominance corner. Let s be a dominance corner for S and assume s is dominated by all the other points. The method considers for each point v of S, the smallest response containing points s and v. The points contained in this response are the points of S dominated by v. For example, in the point set of Figure 4.2, we have that point s is a dominance corner. Also,

the smallest response containing s and v_3 is $\{s, u_3, u_4, v_3\}$, which implies that the points dominated by v_3 are s, u_3 and u_4.

Points in Different Rows and Columns. Consider two points, p, and q. For some placement of these points into regions, namely when they are in regions in different rows and columns, we can immediately decide their horizontal and vertical order and thus whether they are in a dominance or anti-dominance relation. For example, if p is in BL and q is in MM, MR, TM, or TR, then we have that q is above and to the right of p and thus dominates p. Also, if p is in BM and q is ML or TL, then we have that q is above and to the left of p and thus q anti-dominates p. Similar considerations hold for other placements of p and q in different rows and columns.

Points in Different Regions in Same Row or Column. Consider now the case when p and q are in different regions that share the same row or column. In this case, we know one of the horizontal or vertical ordering of the points, but not the other. Let p be in TL and q be in TR. We have that p is to the left of q. We can use our building block method applied to the points in the top row and their anti-dominance corner right to determine whether p and q are in anti-dominance relation. If they are not, given that p is to the left of q, we conclude that q dominates p. The same reasoning holds when p is in TL and q is in TM and, more generally, by symmetry, for p and q in contiguous regions of the same row or column.

Points in Same Region. We now turn to the case when p and q are in the same region. Here, we need to take into account the configurations of the extreme points. We distinguish the cases when top is to the left bottom (Figure 4.3a) and top is to the right of bottom (Figure 4.3b). It is worth noting that we can distinguish these two cases from the response set only if there is at least a point in the middle column. Otherwise, top and bottom are an antipodal pair (Definition 13 and Proposition 4.3.1).

In the case of Figure 4.3a, each region is included in a group of regions that has a dominance corner and another group of regions that has an anti-dominance corner. For

example, suppose p and q are in TL, TM, ML, or MM. We have that left is a dominance corner for the top two rows and bottom in an anti-dominance corner for the left two rows. Applying our building block method to these two groups of regions, we determine whether p and q are in dominance or anti-dominance relation. In the case of Figure 4.3a, we can use the same approach for all regions except MM.

To deal with the remaining case of p and q within region MM in the configuration of Figure 4.3b, we observe that using dominance corner top or bottom, we can determine if p and q are in dominance relation. If so, we are done, else, we find the extreme points of MM and apply the order reconstruction algorithm recursively to the points within this region.

4.4.1 Proof of Theorem 4.3.3

Now that we have introduced the notion of partitioning the domain, we present the proof for Theorem 4.3.3 below.

Proof. Let D be a database and let left, right, top, and bottom be its four extreme points. Without loss of generality, these points must take one of the two configurations pictured in Figure 4.3. Note, any point's relative order can be determined if it is in a dominance relation with one point and in an anti-dominance relation with another point. If a point is not in such a relation, then we argue that the three transformations yield all databases equivalent to D with respect to the response set.

Case 1: If top and bottom are antipodal, we have the configuration of Figure 4.3a or Figure 4.3b with an empty middle column and the ordering of all pairs of points is determined with the exception of the antipodal pair (Transformation 2).

Case 2: If top and bottom are *not* antipodal, we have two subcases.

Case 2a: If top anti-dominates bottom, we have the configuration of Figure 4.3a where the ordering of all pairs of points is determined.

Case 2b: Else, top dominates bottom and we have the configuration of Figure 4.3b, where the

ordering of all pairs of points is determined except for pairs in MM. If $\mathsf{MM} = \emptyset$ or has a single point, we are done. Else, let C be the subset of points of MM are not in anti-dominance relation with a point of D not in MM. We have that all the remaining points of MM have their ordering determined. Also, C comprises one or more components and/or close pairs whose ordering can be changed by means of Transformations 1 and 3.

Now, let us show that there are no other possible transformations that change the order of some pair of points a, b in C, while leaving $\mathsf{RS}(D)$ the same. If b minimally dominates a, there exists no response in $\mathsf{RS}(D)$ that contains right and a without b. Any such transformation would result in one of the following changes: (i) a dominates b, (ii) a anti-dominates b, (iii) b anti-dominates a and (iv) a and b are collinear. If (i), (ii) or (iii), then there would exist a response in $\mathsf{RS}(D)$ that contains right and a, but not b, which would result in a different response set. Thus, the transformation would make a and b be collinear. This is possible only if the corresponding sets X_1, X_2 and X_3 shown in Figure 4.1c are empty. As b minimally dominates a, X_3 must be empty. Suppose there is some point $c \in X_1$, then there is a response that contains a and c without b and a response that contains b and c without a. If a and b were collinear, one of those responses becomes impossible, modifying the response set. A similar argument can be made about X_2. We conclude a and b are a close pair and that we are applying Transformation 3.

Alternatively, if b minimally strictly anti-dominates a, there exists a response r_1 that contains right and a without b and a response r_2 that contains right and b without a. The transformations would result in one of the following: (i) a dominates b, (ii) b dominates a, (iii) a anti-dominates b and (iv) a and b are collinear. In (i), (ii), or (iv) one of r_1 or r_2 would not exist, resulting in a different response set. What is left is case (iii), which implies that the anti-dominance relationship is flipped by applying Transformation 1. □

4.5 Order Reconstruction

We show that the adversary can reconstruct the order of all records in the database (up to equivalent orders) by using the response set. The order reconstruction (OR) algorithm has the following steps:

(1) Find the extreme points of the database. (Algorithm 11)

(2) Find the first antichain of the database, which contains all points that do not dominate any point and generate the dominance graph of the database. (Algorithm 12)]

(3) Find all antichains in the dominance graph. (Algorithm 13)

(4) Build the anti-dominance graph from antichains (Algorithm 14)

(5) Use the dominance and anti-dominance graphs to find any antipodal pairs (Proposition 4.3.1), close pairs (Proposition 4.3.2) and reflectable components. (Proposition 3.3.2). (Algorithm 15)

Note that this attack achieves FDR when the horizontal and vertical projections of the points are dense.

4.5.1 Preliminaries

Our OR attack requires computing the IDs of the points dominating a point in antichain A_0. Algorithm 10 (DominanceID), takes as input the response set $\mathsf{RS}(D)$ of a database D and the ID a of some point with height 0, and outputs the set of identifiers of points that dominate $D[a]$.

We now describe how given the response set $\mathsf{RS}(D)$ and the ID a of a point with height 0, we compute the full set of IDs of points that dominate $D[a]$. Let $a, b \in [R]$ be the IDs of two points in D. Algorithm Boxes, takes as input a pair (a, b) and returns the following responses of $\mathsf{RS}(D)$ (see Figure 4.4):

Algorithm 8 Boxes(a, b)

1: Let $S_{a,b}$ be the smallest response in $\mathsf{RS}(D)$ containing a and b,
2: Let $L = D$ and $P_{b,a}$, $P_{a,b}$ be empty lists
3: **for** $p \in L$ **do**
4: **if** $\nexists r \in \mathsf{RS}(D)$, s.t. $p, b \in r$ and $a \notin r$ **then**
5: Add p to $P_{a,b}$
6: **if** $\nexists r \in \mathsf{RS}(D)$, s.t. $p, a \in r$ and $b \notin r$ **then**
7: Add p to $P_{b,a}$
8: **Return** $P_{b,a}$, $S_{a,b}$ and $P_{a,b}$

- $S_{a,b}$: minimal response containing a and b.
- $P_{a,b}$: D minus the maximal responses containing b but not a; i.e., set of points p such that every response containing b and p contains also a.
- $P_{b,a}$: D minus the maximal responses containing a but not b; i.e., set of points p such that every response containing a and p contains also b.

Given a pair of IDs (a, b), there are at most two distinct maximal responses containing a but not b (or b but not a). These responses comprise the points in the maximal horizontal and vertical strips of the domain that contain a but not b (or b but not a). If a and b share the same horizontal or vertical coordinate, only one of the above strips is nonempty. Algorithm 10 (DominanceID) uses Boxes to determine if top dominates a. If yes, then we return the minimal response containing a, top and right. Else top must strictly antidominate a. Let S be the smallest response containing a, top and right and let M be the smallest response containing

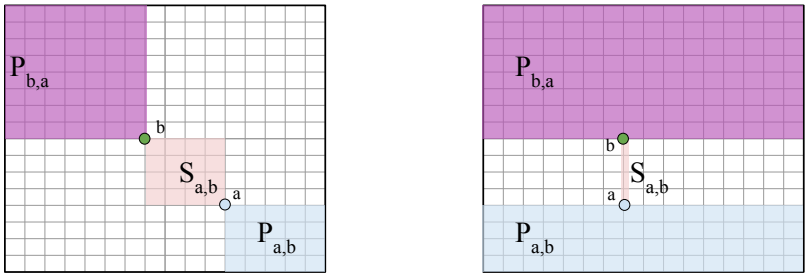


Figure 4.4: Sets output by Algorithm 8 for $a, b \in \mathcal{D}$, when b strictly anti-dominates a (left) and they are co-linear (right).

Algorithm 9 $\mathsf{Edges}(S, \mathsf{RS}(D))$

1: Let RS' be the set of responses that contain only points in S
2: Let L be the largest response in RS'
3: Let S_1 be the 2^{nd} largest response in RS'. $E_1 = L - S_1$.
4: Let S_2 be the 2^{nd} largest response containing E_1. $E_2 = L - S_2$.
5: Let S_3 be the 2^{nd} largest response containing E_1 and E_2. If S_3 exists, $E_3 = L - S_3$.
6: Let S_4 be the 2^{nd} largest set containing E_1, E_2, and E_3. If S_4 exists, $E_4 = L - S_4$.
7: **Return** E_1, E_2, E_3, E_4

a and top. It is clear that $S - M$ contains all IDs of points that strictly dominate a. To find the IDs of points that are colinear with a, we run Edges with $M - \{a\}$ as input; the IDs of points that are colinear with a must be one of the edges in the output. In particular, the colinear points must be $p \in E$ such that E is the edge not containing top, left, or any element of A_0. And so the algorithm outputs $(S - M) \cup E$.

4.5.2 Find Extreme Points

The first step is to identify at most four identifiers of points with extreme coordinate values. Specifically, we wish to find identifiers of points $\mathsf{left}, \mathsf{right}, \mathsf{top}$ and bottom such that for all $p \in D$ the following hold: (1) $\mathsf{left}_0 \leq p_0 \leq \mathsf{right}_0$ and $\mathsf{bottom}_1 \leq p_1 \leq \mathsf{top}_1$, and (2) $p \not\preceq \mathsf{left}, \mathsf{bottom}$ and $\mathsf{top}, \mathsf{right} \not\preceq p$. Note that since no points in D are dominated by left and bottom, then their height is 0 and are thus a subset of A_0 in the canonical antichain partition of D. These points give a starting point for computing the rest of A_0. We recover these extremal points by calling Edges (Algorithm 9).

Our approach for finding such a subset of identifiers is as follows. Let L and S_1 be the first and second largest responses in $\mathsf{RS}(D)$, respectively. Then $E_1 = L - S_1$ must correspond to the IDs of points that are extreme in some coordinate. To find the IDs of points that are extreme in some other coordinate, find the second largest response S_2 that contains E_1, and then compute $E_2 = L - S_2$. By extending this process, we find all points with extremal coordinates. It remains to find the correct point within each set E_i. Suppose E_1 and E_2 are the left and bottom edges, respectively. By finding $a, b \in [R]$ such that the smallest response

Algorithm 10 DominanceID$(a, \mathsf{top}, \mathsf{left}, \mathsf{right}, \mathsf{RS}(D))$

Let S_1 be the smallest response that contains left, top and right.
Let S_2 be the smallest response that contains s_1, top and right.
Let M be the smallest response that includes s_1 and top
for $p \in M$ **do**
    if $p \in M - S_2$ **then**
        $P_{p,\mathsf{top}}, S_{p,\mathsf{top}}, P_{\mathsf{top},p} = \mathsf{Boxes}(\mathsf{top}, p)$
        $S = P_{p,\mathsf{top}} \cup S_{p,\mathsf{top}} \cup P_{\mathsf{top},p}$
        if $\mathsf{left}, \mathsf{right} \in S$ **then**
            Return S_2 // a and top are collinear
        else if $\mathsf{left} \in S$ **then**
            Return S_2 // top dominates a
        else if $\mathsf{right} \in S$ **then**
            // top anti-dominates a
            $E = \mathsf{Edges}(M - \{a\}, \mathsf{RS}(D))$
            $S_2 = S_2 - M$
            Add all p in an edge in E not containing top or $a' \in A_0$ to S_2.
            Return S_2
    else if $p \in M - S_1$ **then**
        $P_{p,a}, S_{a,p}, P_{a,p}\mathsf{Boxes}(a, p)$
        $S = P_{p,a} \cup S_{a,p} \cup P_{a,p}$
        if $\mathsf{left}, \mathsf{right} \in S$ **then**
            Return S_2 // a and top are collinear
        else if $\mathsf{right} \in S$ **then**
            Return S_2 // top dominates a
        else if $\mathsf{left} \in S$ **then**
            // top anti-dominates a
            $E = \mathsf{Edges}(M - \{a\}, \mathsf{RS}(D))$
            $S_2 = S_2 - M$
            Add all p in an edge in E not containing top or $a' \in A_0$ to S_2.
            Return S_2
Return S_2

containing a and b contains no other edge points, then $D[a]$ and $D[b]$ must not be dominating any other points in D. Hence $\mathsf{left} = D[a]$ and $\mathsf{bottom} = D[b]$. Similarly for the identifiers of top and right.

Without loss of generality, we assume that right dominates left. Algorithm FindExtremePairs is inspired by [43] and the pseudocode can be found in Algorithm 11.

Lemma 4.5.1. *Let D be a database with R records and let $\mathsf{RS}(D)$ be its response set. Algorithm 11* (FindExtremePairs) *returns all configurations of extreme points* (left, right, top, bottom) *such that no points are dominated by* left *and* bottom*, and no points dominate* right *and* top *in $O(R^2|\mathsf{RS}(D)|)$ time.*

Algorithm 11 FindExtremePairs(RS(D))

Input: Response set RS(D) of database D

1: $E_1, E_2, E_3, E_4 =$ Edges(D, RS(D))
2: Let PossibleConfigs be all possible combinations of E_1, E_2, E_3 and E_4 into LeftE, RightE, TopE, BottomE.
3: Initialize empty dictionary config.
4: **for** LeftE, RightE, TopE, BottomE in PossibleConfigs **do**
5: **for** $E_1, E_2 \in$ {LeftE, BottomE}, {RightE, TopE} **do**
6: **for** $a, b \in E_1 \times E_2$ **do**
7: **if** the smallest response in RS(D) that contains a and b does not contain any other element of E_1 or E_2 **then**
8: Add a, b to config under their corresponding key left, right, top, or bottom.
9: Return to line 5.
10: Add config to PosExtremes.
11: Return PosExtremes

Proof. We first show that Algorithm 9 returns the correct edges i.e. the sets E_i for $i \leq 4$ contain IDs of all points with an extreme coordinate value. Note that the second largest response in RS(D) must exclude the ID of some extreme point p. For a contradiction, suppose p is not extreme. Then we could minimally extend the query to include p and the resulting query would have a response strictly larger than the original query and strictly smaller than $[R]$ since it is not extreme, hence a contradiction. Now consider the second largest response containing the ID of p. The remaining ID(s) must correspond to points with an extreme coordinate value in another direction, else we could minimally extend the query to include the non-extreme point(s). By extending this reasoning, we recover the IDs of all points with an extreme coordinate.

In Algorithm 11, line 2 stores the at most 4! assignments of the E_i to LeftE, RightE, TopE, and BottomE. The for loop on line 4 then iterates through each possible assignment to identify the correct IDs within each edge set. We want to find the IDs for the left-most point, a, and bottom-most point, b, such that no points are dominated by $D[a]$ or $D[b]$. This corresponds to finding $a \in$ LeftE and $b \in$ BottomE such that the minimal response containing them contains no other extreme points. Suppose for a contradiction that some edge point c was dominated by either a or b, then the minimal query must also contain c. A similar argument holds for the top-most and right-most points.

The algorithm terminates in $O(R|\mathsf{RS}(D)|)$ time. It takes $O(R^2 \cdot |\mathsf{RS}(D)|)$ time to find the edges. Then, we iterate through pairs of edges and look through $\mathsf{RS}(D)$ to find a smallest response. □

4.5.3 Generate Dominance Graph

This step takes as input the response set $\mathsf{RS}(D)$ and some configuration config given by running FindExtremePairs (Algorithm 11) on $\mathsf{RS}(D)$, and outputs a dominance graph G of D. We first compute all IDs of points with height 0. These are the sinks of G. Let left, right, and bottom be given by config. All points not dominated by left and bottom must be contained in the minimal query containing them.

Then for each $a \in A_0$ we build a subgraph of the dominance graph on a and all IDs that dominate a. We use DominanceID (Algorithm 10) to compute this set of IDs. We initialize subgraph $G_a = \{a\}$ and then extend the graph by finding the next smallest response resp containing a, that also contains some ID v not yet added to the graph. Since resp is minimal, then v must dominate everything in the response. Moreover, v must minimally dominate all IDs that are sinks in the current G_a *and* are contained in resp. We add (t, v) to G_a for all sinks t of G_a contained in resp.

Once graphs G_a for $a \in A_0$ have been computed, we take their union, $G = \cup_a G_a$, as the dominance graph and return G and A_0.

Lemma 4.5.2. *Let D be a database with R records, $\mathsf{RS}(D)$ be its response set, and* config *the correct configuration output by* FindExtremePairs *(Algorithm 11) on $\mathsf{RS}(D)$. Given $\mathsf{RS}(D)$ and* config*, Algorithm 12* (DomGraph) *returns the dominance graph of the points in D in $O(R^3|\mathsf{RS}(D)|)$ time.*

Proof. Let left, right, top and bottom be the points defined by config. Without loss of generality, assume that right dominates left and bottom. We first show that lines 2 to 7 find a set of IDs of points that are not dominating any point in D (i.e. a minimal antichain A_0 of D up to

Algorithm 12 DomGraph(RS(D), config)

```
1: // Find antichain-0. We assume right dominates left.
2: Let small be the smallest response containing left and bottom.
3: Let A_0 = small
4: for p ∈ small do
5:     Let S be the smallest response that contains right and p.
6:     Q = (S ∩ small) − {p}
7:     A_0 = A_0 − Q
8: // Find dominance graph.
9: Let G be an empty graph
10: for each a ∈ A_0 do
11:     G_a = (V, E) such that V_a = {a} and E_a = ∅.
12:     S = DominanceID(a, top, left, right, RS(D))
13:     Let R_S ⊆ RS(D) comprise the responses of size at least 2 that contain a and only other IDs in S.
14:     for resp ∈ R_S by increasing size do
15:         if ∃v ∈ resp such that v ∉ G_a then
16:             Add vertex v to G_a
17:             for each t of resp such that t is a sink of subgraph of G_a that contains only points in resp do
18:                 Add edge (t, v) to G_a.
19: G = ∪_{a∈A_0} G_a, and remove any transitive edges
20: Return G, A_0
```

rotation/reflection). By Algorithm 11, no point is dominated by either left or bottom. Let S be the smallest response in RS(D) containing left and bottom. All points *not* dominated by left and bottom must be in S, and thus $A_0 = S$.

By assumption, right must dominate all points with IDs in S. Let p be a point with ID in S and consider the response T of query (p, right). If there is a point q with ID in S such that $p \preceq q$, then its ID must also be in response T. In line 6 we find the set Q of all such IDs and delete Q from A_0. Since the for loop on line 4 iterates through all IDs in S, and deletes the IDs of all points that must dominate at least one other point in S, then at the end of the loop A_0 must be the set of all points not dominating any other point.

On lines 10 to 18, we construct the dominance graph. Let S be the IDs output by DominanceID(RS(D), a) for some $a \in A_0$. Note that $S - \{a\}$ corresponds to the IDs of all records that dominate $D[a]$. The for loop starting on line 14 correctly builds the dominance subgraph on all IDs in S. We show that the following loop invariant is maintained: at the end of iteration ℓ (1) no point with ID in $S \setminus V(G_a)$ is dominated by a point with a vertex in

G_a and (2) if i and j are in $V(G_a)$ and $D[j]$ minimally dominates $D[j]$, then edge (i, j) is in G_a. At the start $G_a = \{a\}$; this is correct since $a \in A_0$ and A_0 is the set of IDs of points that do not dominate any other point. Assume that at iteration ℓ the invariant holds. Find the next smallest response T that contains a and only other IDs in S. If T contains v not in G_a then add it to G_a. (1) holds since no point in $S \setminus V(G_a)$ dominates $D[v]$, otherwise it would be contained in T and we could form a strictly smaller response contradicting the minimality of T. For each sink $t \in G_a$ such that $t \in T$ we add (t, v) to G_a. (2) holds since $D[v]$ must dominate all points with IDs in $T \cap V(G_a)$ and must minimally dominate all sinks t in G_a that are contained in T. Suppose there is some ID j in $V(G_a)$ that is minimally dominated by v but is not a sink. Then this would violate the correctness of G_a at the end of iteration ℓ and hence this cannot happen.

Putting it all together, we want to show that taking the union of all G_a gives us the complete dominance graph G. Let $p, q \in D$ be any points such that $p \preceq q$. By correctness of A_0, there exists some $a \in A_0$ such that $D[a] \preceq p, q$, and thus p and q are contained in the minimal query of a, right, and top. By the correctness of G_a, then an edge from the IDs of p to q must be added when constructing G_a. Since every dominance edge is added to a graph G_a of some a, then taking the union over all G_a gives the complete dominance graph of D. The Algorithm terminates in $O(R^3|\mathsf{RS}(D)|)$ time. It takes $O(R \cdot |\mathsf{RS}(D)|)$ time to find the first antichain. Then, Algorithm 10 takes $O(R^2 \cdot |\mathsf{RS}(D)|)$ and may be run R times. □

4.5.4 Construct Antichains

Given A_0, we now compute the entire canonical antichain partition of D. We explain how to find the partition $\mathcal{A} = (A_0, \ldots, A_L)$ such that L is the maximum height of any element in D. Computing each A_i is equivalent to finding the set of elements whose maximum length path in G from any $a \in A_0$ has length i. Thus, for each $p \in G$ we compute the longest path in G from any $a \in A_0$ to p and then add p to the correct partition in $\mathcal{A}$. Lastly, order the

elements in each antichain $A \in \mathcal{A}$ such that, without loss of generality, for any pair of ordered elements c and c', c' antidominates c i.e. $c \preceq_a c'$. If $|A| \leq 2$ we are done. Else we compute all responses that contain exactly two elements in A. If such a response exists for a pair $c, c' \in A$ then we can infer that there exists no $c'' \in A$ such that $c \preceq_a c'' \preceq_a c'$. Thus we may use these responses to determine the ordering of the elements in A such that any element must anti-dominate all previous elements in the ordering.

Lemma 4.5.3. *Let D be a database and $\mathsf{RS}(D)$ be its response set. Given $\mathsf{RS}(D)$, a dominance graph G of D, and the minimal antichain A_0, Algorithm 13 (FindAntichains) returns a dictionary* Antichains *such that* Antichains$[i]$ *contains an ordered list of all IDs at height i in $O(R^2|\mathsf{RS}(D)|)$ time.*

Proof. Let A_0 be the set of IDs of points with height 0. We argue that the height of $p \in V$ is given by the maximum length of a path from a to p over all $a \in A_0$. Fix some $p \in V$ and suppose that the maximum length of any path from the vertices in A_0 to p is ℓ, and let there be such a maximal path from some $a \in A_0$ to p. By correctness of Algorithm 12, the path from a to p in G corresponds to a chain in database D. Thus the height of p is $\geq \ell$. Suppose for a contradiction that p has height $\ell' > \ell$; By definition of height there must exist a chain $C \subseteq D$ of size ℓ' with p as the maximal element. Let $c_1 \preceq c_2 \preceq \cdots \preceq c_{\ell'}$ be the elements of C. We have that c_{i+1} must minimally dominate c_i, otherwise we could could extend the chain from a to p to have length greater than ℓ'. By correctness of G, each edge (c_i, c_{i+1}) must be in G. Hence the length of the longest path from a to p in G is $\ell' > \ell$, a contradiction. Thus the height of p is given by the length of the longest path from a to p over all $a \in A_0$. Let L be the number of partitions in the canonical antichain partition of D. We have shown that Algorithm computes the partition $\mathcal{A} = (A_0, \ldots, A_L)$ correctly. Let $a_1, \ldots, a_m$ be elements of a partition $A \in \mathcal{A}$. We show that Algorithm 13 correctly computes an ordering of $a_1, \ldots, a_m$ i.e. a $a_{\gamma_1}, \ldots, a_{\gamma_m}$ such that $\gamma_i = 1, \ldots, m$ and for all j either $a_{\gamma_j} \preceq_a a_{\gamma_{j+1}}$ or $a_{\gamma_{j+1}} \preceq_a a_{\gamma_j}$. If $|A| < 3$ then we are done. $|A| \geq 3$ then on line 12 we compute all responses in $\mathsf{RS}(D)$ that

Algorithm 13 FindAntichains($\mathsf{RS}(D), G, A_0$)

```
// Find antichains.
(V, E) = G, Antichains = {}, Antichains[0] = A_0
Compute longest paths∈ G from all a ∈ A_0 to all points in D.
L = 0
for each p ∈ V do
    Let ℓ be the length of the longest path to p from any a ∈ A_0.
    Add p to Antichains[ℓ]
    L = max(L, ℓ)
// Order the points of Antichains[i].
for i = 0, ··· , L do
    if |Antichains[i]| > 3 then
        Let S be all responses in RS(D) that contain exactly two elements of Antichains[i] (and perhaps other points)
        Remove all p ∉ Antichains[i] from S and make S a set.
        Order Antichains[i] such that pairs of consecutive points are responses in S.
Return Antichains
```

contain exactly two elements in A and denote this set as S. A response containing exactly two elements $a, a' \in A$ exists only if a minimally anti-dominates a' (or vice versa). Next we delete all $p \in D - A$ from responses in S and make it a set. Let $\{a, a'\}$ be an element of the resulting set S. Without loss of generality, suppose a' minimally anti-dominates a. Suppose that there exists another set $\{a', a''\} \in S$. Then by transitivity a'' must minimally anti-dominate a'. We can thus "order" the elements in A by finding consecutive pairs of points in the responses.

This Algorithm terminates in $O(R^2|\mathsf{RS}(D)|)$ time, as it takes $O(R^2)$ time to find the longest paths in G and $O(R^2|\mathsf{RS}(D)|)$ to order the antichains. □

4.5.5 Generate Anti-Dominance Graph

The next step is to take the response set $\mathsf{RS}(D)$, the dominance graph G, and the canonical antichain partition Antichains and construct the corresponding anti-dominance graph. There are three major steps that we must take: (1) fix the antichain orientations so that they are lined up correctly, (2) add any edges between IDs of different antichains that are in an anti-dominance relationship, and (3) identify colinearities.

First we iterate through Antichains; At iteration i, we look at Antichains$[j]$ for all $j < i$ until we find an edge (c_1, c_2) in G such that $c_1 \in$ Antichains$[j]$ and $c_2 \in$ Antichains$[i]$. If there is another edge (c_1', c_2') in G with $c_1' \in$ Antichains$[j]$ and $c_2' \in$ Antichains$[i]$, then we check if the edges in the antichains i and j are consistent. E.g, if the orderings are (c_1, c_1') and (c_2', c_2) in Antichains$[j]$ and Antichains$[i]$, respectively, then we flip Antichains$[i]$.

Once the chains are fixed, we add edges for anti-dominance relationships. We iterate through Antichains$[i]$ and Antichains$[j]$ for $i < j$ and look at each pair of elements a_i, a_j such that $a_i \in$ Antichains$[i]$ and $a_j \in$ Antichains$[j]$. For each a_i and a_j we compute all their successors and all predecessors in G. If there exists a path from some successor of a_j to some predecessor of a_i, then we add (a_j, a_i) to G'. Similarly, if there exists a path from some predecessor of a_j to some successor of a_i, we add (a_i, a_j) to G'.

The last thing that remains is to identify colinearities. For each edge (q, p) in G' find the smallest response S containing q and p. If there exists some $k \in S$ such that k and p are not connected in G', then they must be colinear and so we add (k, p) to G'. We similarly check if there exists a colinearity between k and q and add those edges to G'. The final step is to remove all transitive edges in G' (if they exist) to keep only minimal anti-dominance relationship and return the anti-dominance graph G'.

Lemma 4.5.4. *Let D be a database and $\mathsf{RS}(D)$ be its response set. Given $\mathsf{RS}(D)$, the dominance graph G of D, and the ordered antichains of D, Algorithm 14 returns the anti-dominance graph of D in $O(R^3|\mathsf{RS}(D)|)$.*

Proof. The antichains returned by Algorithm 13 may have inconsistent direction. The first step of Algorithm 14 is to fix their orientation. We assume that the first antichain, A_0, has the correct orientation. Then, we find the first element of A_0 that has a dominance edge to a point in A_1, the second antichain. Let that edge be $(c_1, c_2), c_1 \in A_0, c_2 \in A_1$. If there are multiple options for c_2, we pick the smallest one in order. Note that each member p of antichain i must have a dominance edge with some member q of antichain j, $j < i$. Otherwise,

Algorithm 14 AntiDomGraph(RS(D), G, Antichains)

Initialize empty graph G'
// Fix chain orientation
for $i \in [1, |\text{Antichains}|]$ **do**
    Add an edge in G' between consecutive points in Antichains$[i-1]$
    Find $(c_1, c_2) \in G$, where c_1 is the first point in Antichains$[k], k < i$ in an edge with a point from Antichains$[i]$. If there are multiple options for c_2, pick the smallest one in order.
    if $\exists (c_1', c_2') \in G$, for a point $c_1' \in$ Antichains$[k], k < i$, which is after c_1 in order, and $c_2' \in$ Antichains$[i]$, which is before c_2 in order, and there is no path from c_1' to c_2 in G **then**
        Flip the order of Antichains$[i]$
Add an edge in G' between consecutive points in the last antichain
// All chains are fixed; Now add edges between them.
for $A_i =$ Antichains$[i]$ and $A_j =$ Antichains$[j]$, such that $i, j \in [|\text{Antichains}|]$ and $i < j$ **do**
    for $a_i \in A_i$ and $a_j \in A_j$ **do**
        if a_i and a_j not connected in G **then**
            Find successors of a_j, $S_j \subseteq A_j$, and all predecessors of a_j, $P_j \subseteq A_j$. Add a_j to S_j, P_j.
            Find successors of a_i, $S_i \subseteq A_i$, and all predecessors of a_i, $P_i \subseteq A_i$. Add a_i to S_i, P_i.
            if $\exists$ path from p to q in G, s.t. $p \in S_j$, $q \in P_i$ **then**
                Add edge (a_j, a_i) to G'
            else if $\exists$ path from p to a_j in G, s.t. $p \in P_i$ **then**
                Add edge (a_j, a_i) to G'
            else if $\exists$ path from p to q in G, s.t. $p \in P_j$, $q \in S_i$ **then**
                Add edge (a_i, a_j) to G'
// Find any collinearities.
// The pseudocode for Boxes can be found in the Appendix.
Let E be an empty list.
for $(q, p) \in G'$ **do**
    $P_{q,p}, S_{p,q}, P_{p,q} = \text{Boxes}(p, q)$
    Let $S = P_{q,p} \cup S_{p,q} \cup P_{p,q}$
    if $\exists k \in S$, where there is no path from k to p in G' **then**
        Add an appropriate edge between k and p in G'
    if $\exists k \in S$, where there is no path from k to q in E **then**
        Add an appropriate edge between k and q in E
Add all edges in E to G'
Remove transitive edges from G'
Return G'

p would be part of some previous antichain. If the order of antichain 1 is wrong, then a point $c_1' \in A_0$ in order before c_1 must have an edge with point $c_2' \in A_1$, in order after c_2. If the chains were correctly ordered that would be impossible as c_2' anti-dominates c_1 and c_1 anti-dominates c_1'. Thus, c_2' cannot dominate c_1'. Thus, Algorithm 13 can correctly orient the second chain given the order of the previous antichains. Maintaining this invariant, Algorithm 13 correctly orients all antichains.

We begin constructing the anti-dominance graph by adding anti-dominance edges between

consecutive pairs of points in each antichain. It remains to add anti-dominance edges between points in different antichains. The algorithm iterates through pairs of chains, and finds points a_i and a_j that are not connected in G and $a_i \in A_i$, $a_j \in A_j, i < j$. Point a_i either anti-dominates a_j or a_j anti-dominates a_i. In order to determine their relationship, we look for a dominance edge between the antichains. If a_j anti-dominates a_i, then all predecessors of a_i are also anti-dominated by a_j and its successors. So, if a predecessor of a_j dominates a successor of a_i. Then a_j must anti-dominate a_i. Similarly, if a successor of a_j dominates a predecessor of a_i, then a_i anti-dominates a_j.

This technique finds only strict anti-dominance edges. It remains to find any collinear anti-dominance edges. Given a pair of points p and p, such that q anti-dominates p, and a point k that is in $\mathsf{Boxes}(p, q)$, k must have an anti-dominance relationship with both. If no such path exists in G', we add appropriate edges depending on which of the Boxes k is in. Note that in some cases, as explained by Proposition 4.3.2, it's impossible to determine all collinearities. Our definition of the anti-dominance graph is that it contains minimal anti-dominance edges. Thus, after we remove any transitive edges, we have generated D's anti-dominance graph.

The algorithm takes $O(R^2|\mathsf{RS}(D)|)$ time: $O(R^2)$ to fix the antichains and add edges between them, and $O(R^3 \cdot |\mathsf{RS}(D)|)$ to run Boxes for any anti-dominance pair. □

4.5.6 Order Reconstruction

We have already given algorithms for computing the extreme points, the dominance graph, the antichains, and the anti-dominance graph. We now put these pieces together to achieve OR of a database D given its response set $\mathsf{RS}(D)$. Algorithm 15 performs OR by taking the following steps. First it runs $\mathsf{FindExtremePairs}$ (Algorithm 11) to compute all candidate configurations of the extreme points. There is a constant number of such configurations and at least one of them corresponds to a correct arrangement of the extreme points in D (up

to rotation/reflection). For each candidate configuration, it then computes the dominance graph using Algorithm 12 (DomGraph) and the anti-dominance graph using Algorithm 14 (AntiDomGraph). Incorrect configurations result in graphs that are either of an incorrect form or result in a pair of dominance and anti-dominance graphs (G, G') such that databases with orders described by (G, G') are not compatible with $\mathsf{RS}(D)$. Algorithm 15 continues to iterate through the configurations until a correct pair of graphs (G, G') is found and returned. Given a response set $\mathsf{RS}(D)$ of some database D as input, Algorithm 15 (OrderReconstruction) is guaranteed to terminate and output a correct graph pair.

Theorem 4.5.5. *Given the response set* $\mathsf{RS}(D)$ *of a 2D database* D *with* R *records, Algorithm 15 (*OrderReconstruction*) returns an* $O(R)$*-space representation of the set* $\mathsf{E_o}(D)$ *of all possible orderings of the points of databases equivalent to* D *with respect to the response set. The algorithm runs in time* $O(R^3|\mathsf{RS}(D)|)$*, which is* $O(R^7)$*.*

Proof. By Lemma 4.5.1, PossibleConfigs has all possible configurations of a given set of extreme points. Thus, at some point we pick the correct config. By Lemmas 4.5.2 and 4.5.4, we know that G and G' return correct weak dominance and anti-dominance graphs. By Proposition 4.3.1, we know that if the smallest response that contains top and bottom is empty, then they are an antipodal pair. Similarly for left and right. We find all such pairs. We iterate though pairs of points and find any that satisfy the close pair requirements from Definition 14, constructing the closePairs set. The anti-dominance graph encodes the components as the connected components of the anti-dominance graph form the flippable components.

By Theorem 4.3.3, given $(G, G', \mathsf{antipodalPairs}, \mathsf{closePairs})$ output by the algorithm, we can construct all members of set $\mathsf{E}(D)$. The first graph we return is sufficient as any other extreme point configurations whose response set matches $\mathsf{RS}(D)$ are either rotations/reflections or contain antipodal pairs. This Algorithm takes $O(R^3|\mathsf{RS}(D)|)$ time, as it takes $O(R^3|\mathsf{RS}(D)|)$ time to run Algorithms 11, 12, 13 and 14. Finding antipodal pairs takes $O(|\mathsf{RS}(D)|)$ and

Algorithm 15 OrderReconstruction(RS(D))

PossibleConfigs = FindExtremePairs(RS(D))
for config $\in$ PossibleConfigs **do**
    G, A_0 = DomGraph(RS(D), config)
    G' = AntiDomGraph(RS(D), G, Antichains(RS(D), G, A_0))
    Let closePairs and antipodalPairs be empty lists.
    Find the smallest response that contains top and bottom. If it contains no other points, then add (top, bottom) to antipodalPairs.
    Find the smallest response that contains left and right. If it contains no other points, then add (left, right) to antipodalPairs.
    for each edge $(b, a) \in G$ **do**
        if (b, a) satisfy Definition 14 **then**
            Add (b, a) to closePairs
    if response set of points with orders (G, G') is RS(D) **then**
        Return $(G, G'$, antipodalPairs, closePairs)

finding close pairs $O(R^3)$. Finally, it takes $O(R^4)$ time to generate and compare the leakage. We can encode graphs G and G' by their linear extensions in linear space, and the sets antipodalPairs and closePairs contain at most $O(R)$ points. □

4.5.7 Experiments

In the previous subsections, we discussed the limitations of OR and described an algorithm that succeeds at OR when given the response set of a database. We now support our theoretical results with experimental results. We have deployed our OR attack on three real-world databases (Table 4.2): California, Spitz and HCUP.

The California Road Network dataset [78] comprises 21,047 road network intersections indexed by longitude and latitude. Our ***California dataset*** is a random sample of 1000 points with coordinates truncated to one decimal place and scaled by a factor of 10. The resulting domain is $[102] \times [102]$. We generated the response set for this dataset and then ran our OR attack (Algorithm 15) on it.

Although, in theory, we only recover the relative orders of all the points, the actual reconstruction leaks additional information about the overall "shape" of the data. For our reconstruction, after finding the order of the points, each point is assigned coordinates

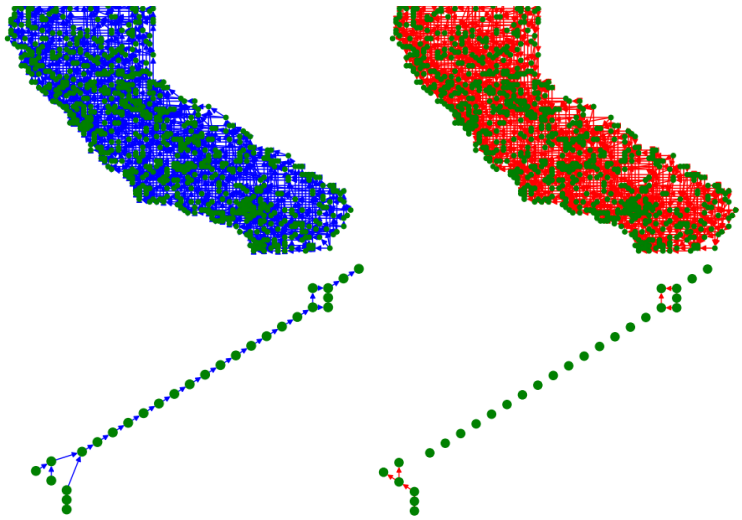


Figure 4.5: Dominance (right) and anti-dominance (left) graphs of the (top) California and (bottom) Spitz datasets.

corresponding to its index in each dimension's ordering. The figure shows each antichain in a different color, illustrating the height increase, as well as an α-shape [41] of the point-set.

Malte Spitz is a German politician who published his phone location for 166 days, between 8/31/2009 and 2/21/2010 [118]. Our ***Spitz dataset*** comprises longitude and latitude information from the first day, truncated to one decimal place and scaled by a factor of 10.

We also ran our order reconstruction attack on the Healthcare Cost and Utilization Project (HCUP) ***Nationwide Inpatient Sample (NIS) 2008 and 2009 medical datasets*** [1], but we are unable to share images of the reconstructions, per the HCUP data usage agreement. The HCUP dataset was used in prior attacks [43, 74, 75]. The reconstructed dominance graph and anti-dominance graph of the California and Spitz datasets are shown in Figure 4.5.

Order reconstruction in two-dimensions is significantly more enlightening than in one-dimension. We conjectured that the geometry of the data is more observable when data is more dense in one or both of the domains. Our results from the California dataset support this: we can clearly see that this location data comes from the state of California. In the Spitz case, we can still recover the shape of the dataset and see that it is a deeply diagonal database with a number of collinearities and reflectable components (Figure 4.5).

4.6 Estimating the Query Density

Recall that the query density, ρ_S, of a set of records S corresponds to the number of unique range queries that contain all records in S. One of the challenges of reconstructing a database D with partial knowledge of the search pattern, is that the adversary can no longer compute the exact ρ values by constructing $\mathsf{RM}(D)$. Thus, the two-dimensional FDR attack [43] no longer applies. To reconstruct with missing queries, we draw inspiration from [73] and use statistical estimators to estimate the ρ values. In Section 4.7 we show how ρ estimates can be used to build a system of non-linear equations whose solution corresponds to an approximate reconstruction of the target database.

Formally, let D be a database of R records and let $M = \{\{(t_1, A_1), \ldots, (t_m, A_m) : A_i \in \mathsf{RS}(D)\}\}$ be a sample (i.e. multiset) of m token-response pairs that are leaked when queries are issued according to an arbitrary distribution. Let $L \subseteq M$ be a subsample of M of size n. Given a sample (multiset) M of m token-response pairs, we show how one may compute the appropriate submultisets $L \subseteq M$ that correspond to the ρ functions of interest. Each of these submultisets is used to approximate the value of its respective ρ value.

Definition 15. *[121] Let L be a subsample and let f_i be the number of search tokens that are observed i times in L. The **fingerprint** of a sample L is the vector $F = (f_1, f_2, ..., f_n)$, where $|L| = n$. We can express the total number of token-response pairs in L as $n = \sum_{i=1}^{n} i f_i$ and the number of observed distinct search tokens as $d = \sum_{i=1}^{n} f_i$.*

To estimate $\widehat{\rho} \approx \rho$, we let L be a submultiset of M comprised of all token-response pairs that contain the identifiers of the points whose ρ value we wish to compute. We then use an estimator to estimate how many unique search tokens are associated with those record identifiers. We describe three such estimators below.

4.6.1 Non-parametric Estimators

Sampling-based estimators have been used in various domains ranging from databases [61] to ecology (e.g. [19,20]). Non-parametric estimators do not require prior knowledge of the query distribution, yet their success hinges upon the underlying distribution from which queries are drawn. Indeed, for skewed distributions, it may be information theoretically impossible to obtain a reasonable estimate. Recently, non-parametric estimators have been used for database reconstruction to estimate the support size of the given conditional probability distribution of a particular record identifier [73].

We have considered the estimators by Chao and Lee [27] and by Shlosser [115], and the jackknife estimators described in [19, 20]. We describe these estimators in more detail below. We initially also considered the Valiant-Valiant estimator [121] as it was used in [73]. However, it did not perform as well in our case.

Chao-Lee. Chao and Lee proposed an estimator that utilizes sample coverage [27]. The sample coverage C of a sample L is the sum of the probabilities of the the token-response pairs that appear in L. Knowledge of C can then be used to estimate $\widehat{\rho}$. Chao and Lee use this approximation in combination with an additive term to correct estimates of data drawn from skew distributions.

Let p_i be the probability that a query sampled from the distribution matches the i-th token-response pair, of the possible $Q = \binom{N_0+1}{2}\binom{N_1+1}{2}$ token-response pairs. Let $\mathbb{1}_L(i)$ be the following indicator function: $\mathbb{1}_L(i)$ equals 1 if the i-th token-response pair is in L and 0 otherwise. The sample coverage C of a sample L is the sum of the probabilities of the the token-response pairs that appear in L: $C = \sum_{i=1}^{Q} p_i \cdot \mathbb{1}_L(i)$. Note that $\hat{C} = 1 - f_1/n$ is a natural estimate for C, which can then be used to estimate $\widehat{\rho} \approx d/\hat{C}$. Thus,

$$\widehat{\rho}_{\text{ChaoLee}} = \frac{d}{\hat{C}} + \frac{n(1-\hat{C})}{\hat{C}} \cdot \hat{\gamma}^2,$$

where $\hat{\gamma}$ is an estimate of the coefficient of variation $\gamma = (\sum_i (p_i - p_{mean})^2/Q)^{1/2}/p_{mean}$ and p_{mean} is the mean of the probabilities $p_1, \ldots, p_Q$.

Shlosser. Shlosser derived an estimator that works well under the assumption that the sample is large and the sampling fraction is non-negligible [115]. We used an implementation of Shlosser Estimator that used a Bernoulli Sampling scheme. This estimator is more effective for skewed distributions.

Let q be the probability with which a token-response pair is included in the sample. In [115], Shlosser derived the estimator

$$\widehat{\rho}_{\text{Shloss}} = d + \frac{f_1 \sum_{i=1}^{n} (1-q)^i \cdot f_i}{\sum_{i=1}^{n} i \cdot (1-q)^{i-1} \cdot f_i}.$$

This estimator rests on the assumption that $q = n/Q$. As [61] notes, the Shlosser estimator further rests on the assumption that $\mathbb{E}[f_i]/\mathbb{E}[f_1] \approx F_i/F_1$ where F_i is the number of tokens that appear i times in entire database; This assumption isn't often satisfied in our setting, but our experiments demonstrate that Shlosser did comparable to Jackknife in various cases.

Jackknife. The jackknife method was introduced as a technique for correcting the bias of an estimator [109]. We use the jackknife estimators described in [19, 20], which have been used for the problem of estimating the number of unique attributes in a relational database [61], in database reconstruction [73], and in biology for the related problem of species estimation. Given a biased estimate, jackknife estimators use sampling with replacement to estimate the bias $bias_{jack}$, and obtain $\widehat{\rho}_{\text{jack}}$.

One can view d as a biased estimate of the true ρ. Given a biased estimate d, jackknife estimators use sampling with replacement to estimate the bias $bias_{jack}$, and obtain $\widehat{\rho}_{\text{jack}} = d - bias_{jack}$. Let d_n denote the number of unique tokens in L and let $d_{n-1}(k)$ denote the number of unique tokens in L when the k-th token-response removed. Note that $d_{n-1}(k) = d_n - 1$ if and only if the k-th pair is unique in L. Let $d_{n-1} = (1/n) \sum_{k=1}^{n} d_{(n-1)}(k)$.

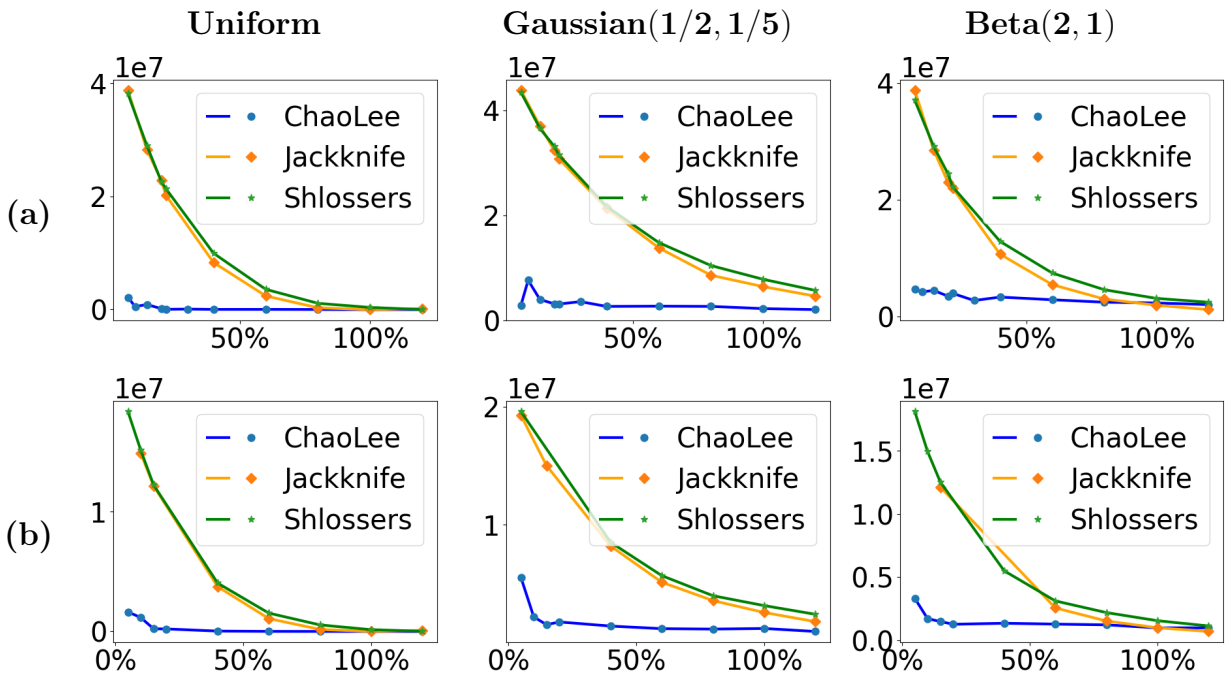


Figure 4.6: MSE of the estimators on the (a) Spitz and (b) 2008 NIS AGE $<$ 18 & NPR datasets over the query ratio.

The first order jackknife estimator is $\widehat{\rho}_{\text{jack}} = d - (n-1)(d_{(n-1)} - d)$. The second order jackknife considers all n samples generated by leaving one pair out, in addition to all $\binom{n}{2}$ generated by leaving two pairs out. This method can be extended to an k-th order jackknife estimators that generates $\sum_{i=1}^{k} \binom{n}{i}$ samples and has bias $O(n^{-k+1})$.

4.6.2 Experiments

We ran our estimators against two datasets with domain sizes 25×25 and 18×33. The first dataset is the first day of the Spitz dataset (described in Section 4.5.7), a dataset deeply diagonal exhibiting numerous collinearities and reflectable components. The second database is the NIS 2008 AGE $\leq$ 18 & NPR database, a fairly dense medical database. We utilized Python library PyDistinct [26] to compute the estimates. They were chosen as they represent two fairly different real-world data distributions. For more information, see Table 4.2.

We tested the robustness of each estimator under the (i) uniform distribution, (ii) Beta(2,1)

distribution and (iii) Gaussian(1/2,1/5) distribution of the queries. Recall that our goal is to estimate the query densities ρ_i for each ID i and $\rho_{i,j}$ for each pair of IDs. Thus, we obtained estimates $\widehat{\rho}_i$ and $\widehat{\rho}_{i,j}$ from the three estimators under the three query distributions and computed the mean squared error (MSE) of such estimates. In Figure 4.6, we plot the MSE of the estimators against the ***query ratio***, which we define as the ratio of the number of queries observed and the total number of possible queries. I.e., if we have observed a queries (including any duplicates) and there are a total of b possible queries, the query ratio is $\frac{a}{b}$. Note that even when this ratio is 1, the adversary most likely has not observed all possible queries. Missing values in the plots of Figure 4.6 are due to failure by the estimators to produce an answer in some cases. Overall, we found that the Chao-Lee estimator consistently performed best, especially for a small query ratio.

4.7 Approx. Database Reconstruction

Our distribution-agnostic attack for ADR assumes the ordering of the points and consists of two parts. As we saw in Section 4.6, non-parametric estimators may perform differently under different query distributions. In our experiments, the Chao-Lee estimator performed the best under all three distributions and we use it to estimate how many query responses contain a point or a set of points. We use these estimates to construct a system of equations, whose solution gives an approximate reconstruction.

4.7.1 Algorithm

We assume knowledge of the ordering (e.g., as given by Algorithm 15). The first step of ADR is to build a system of equations. We know that point p with coordinates p_0, p_1 will be included in $\rho_p = p_0 p_1 (N_0 - p_0)(N_1 - p_1)$ unique responses. The Chao-Lee estimator gives us an estimate, $\widehat{\rho}_p$, of ρ_p. We then construct an equation with unknowns x_p, y_p.

$$\boxed{x_p y_p (N_0 - x_p)(N_1 - y_p) = \widehat{\rho}_p} \tag{4.3}$$

Given a pair of points p, q, where p dominates q, we know that both points are included in $\rho_{p,q} = q_0 q_1 (N_0 - p_0)(N_1 - p_1)$ unique responses. We estimate $\rho_{p,q}$ as $\widehat{\rho}_{p,q}$, and construct an equation with unknowns x_p, y_p, x_q, y_q.

$$\boxed{x_q y_q (N_0 - x_p)(N_1 - y_p) = \widehat{\rho}_{p,q}} \tag{4.4}$$

We build a similar equation from any ordering of p and q, following Equation 4.2. If two points are in both a dominance and anti-dominance relationship, then they must be collinear. We add this constraint to our system. We use the Chao-Lee estimator to approximate the ρ values $(\rho_p, \rho_{p,q})$ from the multiset of responses we have seen. We then construct a first guess for the values of the points using their ordering. Each point p is given coordinates corresponding to its indexes in the first and second dimension. Finally, we find an approximation of the database's point values using a least-squares approach.

Our ADR attack is summarized in Algorithm 16, which takes as input a multiset M of token-response pairs, the ordering G, G' and the domain size (N_0, N_1). It returns a reconstructed point set.

Algorithm 16 $\mathsf{ADR}(M, G, G', N_0, N_1)$

1: Let g be a reconstruction of the point values using G and G', giving each point a value corresponding to its order in each dimension.
2: Create a system of ρ equations for all single points and pairs, including any collinearities, utilizing Equations 4.3 and 4.4.
3: Using the submultiset M of token-response pairs we have observed and the Chao-Lee estimator approximate, the ρ value of each equation, as described in Section 4.6.
4: **Return** a least-squares solution to the system of equations initializing at g

Table 4.2: Real-world datasets used in our experiments.

Dataset	Attributes	# Queries	#Points	Domain
California [79]	LAT & LONG	26532800	1000	102×102
Spitz [118]	LAT & LONG	130500	28	25×25
NIS 2008 [1]	AGE<18 & NPR	80784	355	18×33
	NCH & NDX	663300	529	25×67
	NCH & NPR	158400	574	25×33
NIS 2009 [1]	NCH & NDX	621270	528	27×60
	NCH & NPR	246753	566	27×38
	NDX & NPR	1244310	862	60×38

4.7.2 Datasets and System

The ADR attack assumes the order of the records as an input. For our experiments, we picked the correct order from the results of OR (Algorithm 15). We tested our ADR attack on real world datasets: the California [79] and Spitz [118] location datasets and the HCUP NIS medical datasets [1] described in Section 4.5.7. Due to complexity constraints, we sub-sampled the datasets (resulting domain and more information shown in Table 4.2). We performed one run per experiment and, for each experiment, we sampled queries according to the uniform, Beta$(2, 1)$, and Gaussian$(1/2, 1/5)$ distributions.

Our experiments were run on the Brown University Computer Science Compute Grid, which runs on Intel Xeon and AMD Opteron CPUs and relies on the Oracle Grid Engine to schedule jobs. We implemented our attack in Python 3.7.1. For our experiments, we used PyDistinct [26] to estimate the ρ values and the *least_squares* function from SciPy Optimize [124] to solve our system of equations. The Numpy [62] library was used for general computing.

4.7.3 Accuracy Metrics

We measure the accuracy of the reconstruction with the following four metrics to take into account different characteristics. The mean error is the average distance of a reconstructed point to the original point. We use the ***normalized mean error***, which is obtained by dividing the mean error by $N_0 + N_1$, where $[N_0] \times [N_1]$ is the domain of the database. The ***mean squared error*** is the average squared distance of a reconstructed point to the original point. This widely used error metric (e.g., [73]) gives greater weight to larger errors. The ***Hausdorff distance*** of point sets P and Q, denoted $H(P, Q)$, is a common measure of how far P and Q are from each other. It is defined as $H(P, Q) = \max(h(P, Q), h(Q, P)$, where $h(P, Q) = \max_{p \in P}(\min_{q \in Q} dist(p, q))$. We obtain the ***pairwise relative distance error*** by computing all distances between pairs of original points and between pairs of reconstructed points, calculating the absolute values of the differences of such distances, normalizing by the original distances, and taking the mean. This measure captures the accuracy of the shape of the reconstructed points. For the Hausdorff distance, we use SciPy's [124] implementation of the algorithm in [120]. The other metrics are easily computed.

4.7.4 Experiments

Figure 4.7 shows our reconstructions of the Spitz and California datasets. We cannot present reconstructions of the NIS datasets per the HCUP data usage agreement. In Figure 4.8, and 4.10, we give the accuracy metrics and computational resource usage of our reconstructions for all databases under the different distributions. On the x-axis we show the query ratio, i.e., the number of queries observed by the adversary over the total number of possible queries. Recall that even when this ratio is 1, the adversary most likely has not observed all possible queries as some duplicate queries would typically be issued.

Our attack performs consistently well on both the location and medical datasets under all four metrics and all three query distributions. The four accuracy metrics follow similar trends.

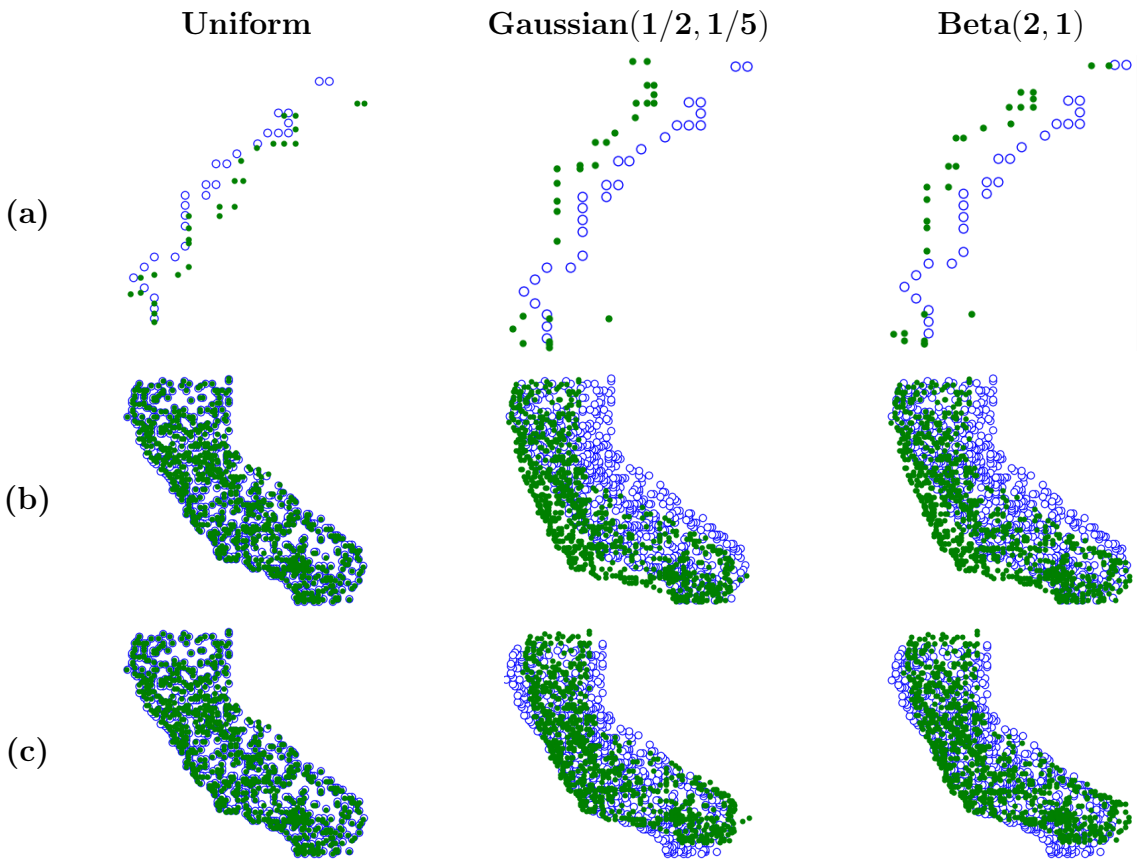


Figure 4.7: Reconstructions generated by our algorithm. Empty blue circles denote original points and filled green circles denote reconstructed points. (a) Spitz dataset with 7% query ratio. (b) California dataset with 4% query ratio. (c) Postprocessing adjustment.

As expected, the accuracy of our reconstruction generally improves with the query ratio. In particular, for the uniform distribution, we already achieve near perfect reconstruction with query ratio around 10%, while for the Beta and Gaussian distributions, there are still errors even at 80% query ratio. Note that the smaller the query ratio is, the higher the variation of accuracy across experiments is, since different query samples vary in usefulness. This is partially due our estimator performing worse under non-uniform distributions and small query ratios (see Figure 4.6).

Our experiments required between a few megabytes of memory to tens of gigabytes for the more computationally intensive ones. Interestingly, the query ratio seems to have a

small effect to the memory required and the size of the database is the deciding factor. The experiments needed from a few CPU seconds to several CPU days to complete. We show the memory and CPU time required by the experiments in the last and second to last columns of Figure 4.8, respectively. Note that our code was not optimized for multi-threaded programming as it was written in Python. In multiple experiments, the CPU usage tends to be high at low query ratios. We believe this phenomenon is caused by inaccurate estimates from the estimators (Section 4.6) that make finding a least squares solution harder.

In Figure 4.10, we show the results of our reconstructions of the NIS 2009 dataset. Overall, the results follow a similar trend to the results in Figure 4.8. There is a decrease in normalized mean error, mean squared error, and pairwise relative distance error, as a larger percentage of queries is observed. We also note that the maximum memory required is fairly constant across all runs.

We ran our experiments on a computing grid that automatically allocated CPUs per experiment. We show in Figure 4.9(a) the histogram of an indicator of the speedup provided by the grid for our experiments. This indicator equals the CPU time divided by the wall-clock time minus 1. Over all experiments, the mean was 0.629, the maximum 4.372, and the variance 0.315, confirming the modest speedup provided by the grid. We infer that our experiments can be reproduced in a computing environment with ≥ 5 CPUs.

4.7.5 Post-processing Adjustment

In a number of datasets, our solution is topologically close to the original data, yet translated. We now explore how to further reduce the reconstruction error. In Figure 4.7b, the shape of California is clear, yet in the Gaussian and Beta cases, the points are shifted towards the bottom right. If we were given the centroid of the original points, we could compare it with the centroid of our solution, and translate all points by their difference, as shown in Figure 4.7c.

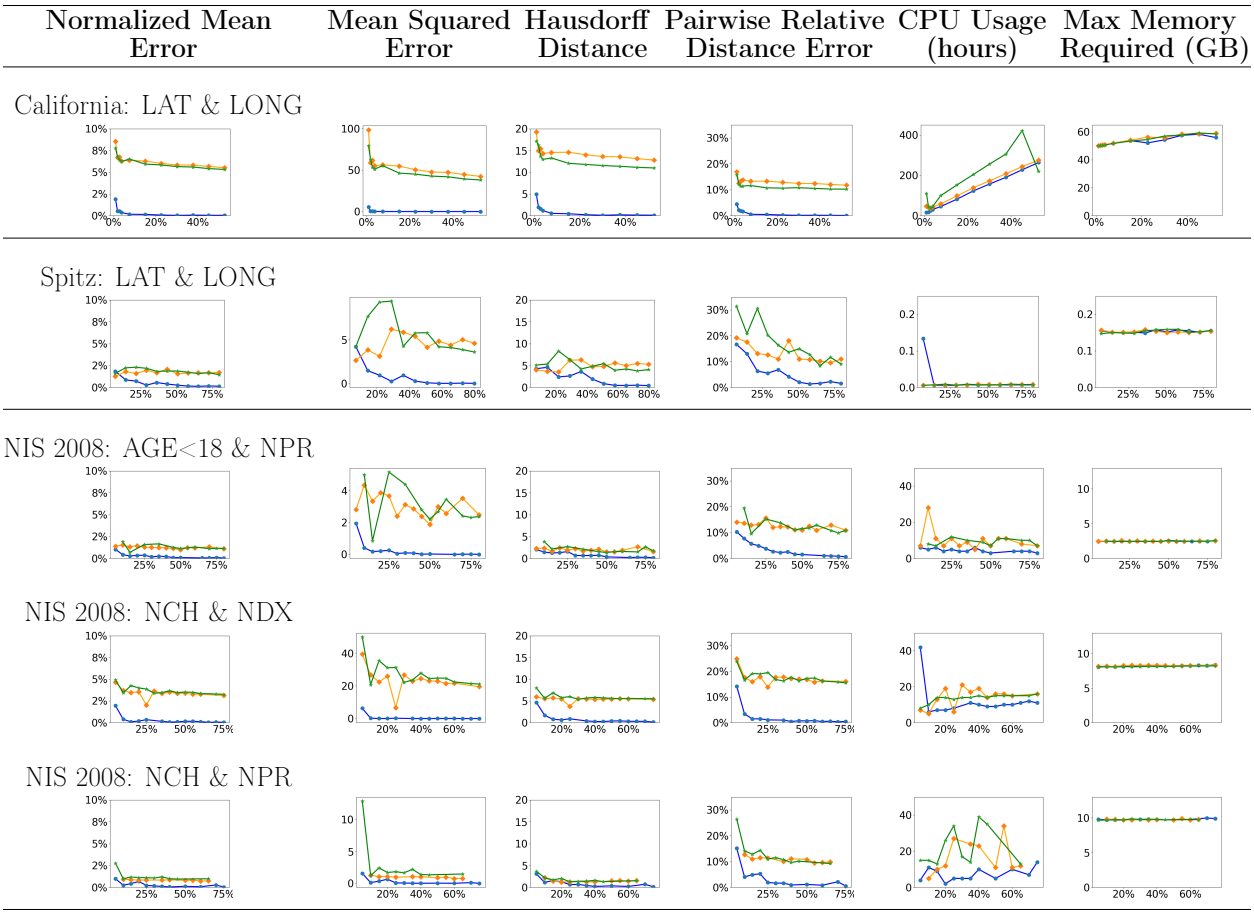


Figure 4.8: Accuracy (measured with the metrics defined in Section 4.7.3) and computational resource usage (CPU time and maximum memory required) of our reconstructions of the California, Spitz and NIS 2008 datasets (see Section 4.7.2) as a function of the query ratio (number of queries observed by the adversary over the total number of possible queries), under the Uniform (blue circle •), Beta (green star ⋆), and Gaussian (orange ♦) query distributions. In Section 4.7.2, we describe each database and its characteristics, including the domain size and the total number of possible range queries.

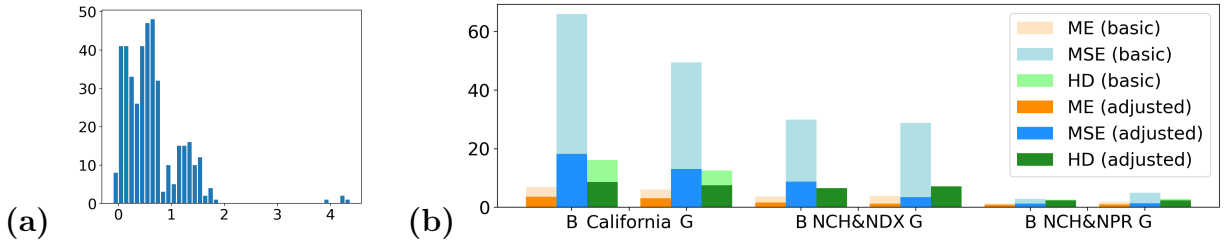


Figure 4.9: (a) Histogram of the grid speedup (CPU time over wall-clock time minus 1) of our experiments. The mean is 0.629, the maximum is 4.3725 and the variance is 0.315. (b) Impact of applying the adjustment technique of Section 4.7.5 to the reconstructions of the California and NIS 2009 NCH & NDX and NCH & NPR datasets for the Beta (B) and Gaussian (G) distributions.

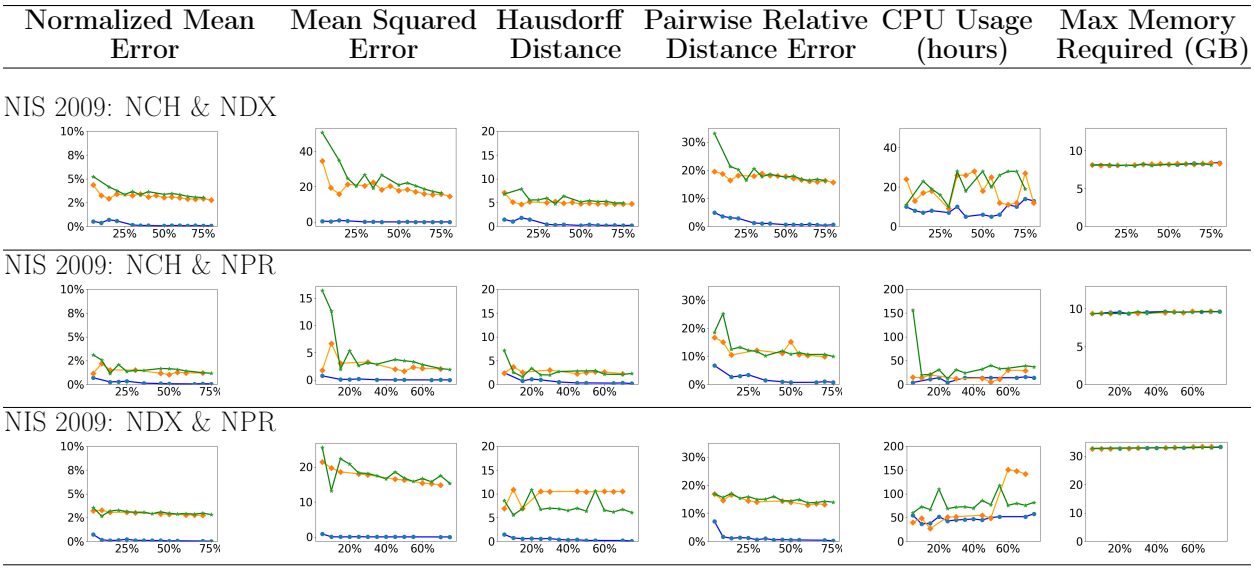


Figure 4.10: Accuracy (measured with the metrics defined in Section 4.7.3) and computational resource usage (CPU time and maximum memory required) of our reconstructions of the NIS 2009 datasets as a function of the query ratio, under the Uniform (blue circle •), Beta (green star ⋆), and Gaussian (orange ♦) query distributions.

We ran this adjustment technique on the reconstructions of the California dataset and NIS 2009 NCH & NDX and NCH & NPR datasets. For the latter, we used the centroids of the corresponding 2008 NIS datasets as proxies for the original centroids. This choice is motivated by the fact that the adversary might have access to the statistics of a related dataset with a similar centroid. We applied the adjustment only to the Beta and Gaussian distributions since our reconstructions under the uniform distribution are already very good. We report in Figure 4.9(b) the variation of the normalized mean error (NME), mean squared error (MSE), and Hausdorff distance (HD) due to our post-processing adjustment. Since we are only translating the points, the pairwise relative distance error does not change. The experiments show that this simple adjustment often significantly reduces reconstruction error.

4.8 Conclusion and Future Work

This chapter presents practical reconstruction attacks from 2D range queries on encrypted databases and furthers our understanding of the intrinsic limitations of such attacks. A first future direction is improving the performance of OR and ADR with advanced data structures and specialized nonlinear solvers. Another direction is exploring partial OR from a subset of the access pattern leakage. All symmetries in two-dimensions occur also in higher dimensions since we can project the points onto a 2D plane. Many of our techniques extend naturally to higher dimensions, including the notion of chains/antichains. It is an open problem whether new symmetries arise in dimensions greater than two.

CHAPTER 5

AN EFFICIENT QUERY RECOVERY ATTACK AGAINST A GRAPH ENCRYPTION SCHEME

This chapter first appeared as a conference publication by the same title with minor modifications in ESORICS 2022 [45]. It is joint work with Kenneth G. Paterson.

5.1 Introduction

Graphs are a powerful tool that can be used to model many problems related to social networks, biological networks, geographic relationships, etc. Plaintext graph database systems have already received much attention in both industry (e.g. Amazon Neptune [5], Facebook TAO [17], Neo4j [97], GraphDB [99]) and academia (e.g. Pregel [86], GraphLab [84], Trinity [112]).

With the rise of data storage outsourcing, there is an increased interest in Graph Encryption Schemes (GES). A GES enables a client to encrypt a graph, outsource the storage of the encrypted graph to an untrusted server, and finally to make certain types of graph queries to the server. Current GES typically only support one type of query, e.g. adjacency queries [29], neighbor queries [29], approximate shortest distance queries [93], and exact shortest path queries [52, 128].

In this chapter, we analyse the security of the GES of Ghosh, Kamara and Tamassia [52] from ASIA CCS 2021. We refer to this scheme henceforth as the ***GKT scheme***. The GKT scheme encrypts a graph G such that when a shortest path query (u, v) is issued for some vertices u and v of G, the server returns information allowing the client to quickly recover the shortest path between u and v in G. The scheme pre-computes a matrix called the ***SP-matrix*** from which shortest paths can be efficiently computed, and then creates an

encrypted version of this matrix which we refer to as the encrypted database (EDB). EDB is sent to the server. At query time, the client computes a search token for the query (u, v); this token is sent to the server and is used to start a sequence of look-ups to EDB. Each look-up results in a new token and a ciphertext encrypting the next vertex on the shortest path from u to v. The concatenation of these ciphertexts is returned to the client and decrypting this sequence reveals the vertices in the shortest path.

The GKT scheme of [52] is very elegant and efficient. For a graph on n vertices, computing the SP-matrix takes time $O(n^3)$ and dominates the setup time. Building a search token involves computing a pseudo-random function. Processing a query (u, v) at the server requires t look-ups in EDB, where t is the length of the shortest path from u to v. Importantly, thanks to the design of the scheme, query processing can be done without interaction with the client, except to receive the initial search token and to return the result. This results in EDB revealing at query time the sequence of labels (tokens) needed for the recursive lookup and the sequence of (encrypted) vertices that is eventually returned to the client.

Ghosh et al. [52] provide a security proof of the GKT scheme in a simulation-based security model that assumes an honest-but-curious (semi-honest) server. The approach identifies a leakage profile for the GKT scheme and formally proves that the scheme leaks nothing more than this. The leakage profile comes in two parts: setup leakage (available to the server upon receipt of the encrypted data structure) and query leakage (that becomes available to the server as it processes each query). Specifically, the query leakage leaks when two queries are equal i.e. the ***query pattern***, the length of the queried path, and how two paths with the same destination intersect.

We exploit the query leakage of the GKT scheme to mount a ***query recovery (QR)*** attack against the scheme. Our attack can be mounted by the honest-but-curious server and requires knowledge of the graph G. This may appear to be a strong requirement, but is in fact weaker than is permitted in the security model of [52], where the adversary can

even ***choose*** G. Assuming that the graph G is public is a standard assumption for many schemes that support private graph queries [52, 94, 111]. This model is perfect for routing and navigation systems in which the road network may easily be obtained online via Google Maps or Waze, but the client may wish to keep its queries private. In such a scenario, the map and traffic information are widely available, but the routing information of individual users is sensitive.

Our attack has two phases. First, it has an offline, pre-processing phase that is carried out on the graph G. In this phase, we extract from G a plaintext description of all its shortest path trees. We then process these trees and compute candidate queries for each query using each tree's canonical labels. A canonical label is an encoding of a graph that can be used to decide when graphs are isomorphic; a canonical label of a rooted tree can be computed efficiently using the AHU algorithm [3]. This concludes the offline phase of the attack. Its time complexity is $O(n^3)$ where n is the number of vertices in G, and matches the run time of our overall attack and the run time of the GKT scheme's setup. Both our attack and the setup are lower bounded by the time to compute the all-pairs shortest paths, which takes $O(n^3)$ time for general graphs [47].

The second phase of the attack is online: As queries are issued, the adversary constructs a second set of trees that correspond to the sequence of labels computed by the server when processing each query i.e. the per-query leakage of the scheme. That leakage is uniquely determined by the search token that initiates the look-up. This description uses the labels of EDB (which are search tokens) as vertices; two labels are connected if the first points to the second in EDB. When an entire tree has been constructed, the adversary can then run the AHU algorithm again to compute the canonical names associated with this ***query tree***. An entire query tree Q can be built when all queries to a particular destination have been issued. In practice, this is a realistic routing scenario where many trips may share a common popular destination (e.g. an airport, school, or distribution center).

By correctness of the scheme, there exists a collection of isomorphisms mapping Q to at least one tree computed in the offline phase. Such isomorphisms also map shortest paths to shortest paths. We thus perform a matching between the paths in the trees from the online phase to the trees in the offline phase. This can be done efficiently using a modified AHU algorithm [3] that we develop and which decides when one path can be mapped to another by an isomorphism of trees. This yields two look-up tables which, when composed, map each path in the first set of trees to a set of candidate paths in the second set. We use the search token of the queries associated with Q to look up the possible candidate queries in the tables computed in the online phase, and output them. The runtime of this phase is $O(n' \cdot n^2)$ where $n' \leq n$ is the number of complete query trees computed in the online phase. The output is guaranteed to contain the correct query.

In general, the leakage from a query can be consistent with many candidates for that query, and the correct candidate cannot be uniquely determined. Graph theoretically, this is because there can be many possible isomorphisms between pairs of trees in our two sets. If we consider the chosen graph setting, it is easy to construct a graph G where, given any query tree Q of G, its isomorphism is uniquely determined and there is a unique candidate for each query of Q, i.e. we can achieve what we call ***full query recovery (FQR)***. For such graphs, the GKT scheme offers almost no protection to queries. In other cases, the query leakage may result in one or only a few possible query candidates, which may be damaging in practice. In order to explore the effectiveness of our attack, we support it with experiments on 8 real-world graphs (6 of which were used in [52]) and on random graphs with varying graph sizes and edge probabilities. Our results show that for the given real-world graphs, as many as 21.9 % of all queries can be uniquely recovered and as many as half of all queries can be mapped to at most 3 candidate queries. Our experimental results show that query recovery tends to result in smaller sets of candidate queries when the graphs are less dense, and that dense graphs tend to have more symmetries and hence result in larger sets of candidate queries.

Note also that our attack is best possible: it always outputs a minimal set of candidates consistent with the query leakage, and the correct query is always included in the set.

We summarize our core contributions as follows:

1. We present the first attack against a GES that supports shortest path queries, and the second known attack against GESs, to our knowledge.

2. We use the GKT scheme's leakage to mount an efficient query recovery attack against the scheme. We explain how, for our real world datasets, the set of all query trees can be recovered with as few as 68.1% of the queries.

3. We make use of the classical AHU algorithm for the graph isomorphism problem for rooted trees and develop a new algorithm for deciding when a path in one tree can be mapped onto a path in another under an isomorphism. Our supporting graph theoretic theorems may be of independent interest.

4. We evaluate our attack against real-world datasets and random graphs.

5. We motivate the need for detailed cryptanalysis of GESs.

5.2 The GKT Graph Encryption Scheme

In this section, we give an overview of the GKT scheme [52] and its leakage.

5.2.1 GKT Scheme Overview

The GKT scheme supports ***single pair shortest path (SPSP)*** queries. The graphs may be directed or undirected, and the edges may be weighted or unweighted. An SPSP query on a graph $G = (V, E)$ takes as input a pair of vertices $(u, v) \in V \times V$, and outputs a path $p_{u,v} = (u, w_1, \ldots, w_\ell, v)$ such that $(u, w_1), (w_1, w_2), \ldots, (w_{t-1}, v) \in E$. We require

that this path is of minimal length in G, i.e. there does not exist a sequence of edges $(u, w'_1), (w'_1, w'_2), \ldots, (w'_{t-1'}, v) \in E$ such that $t' < t$.

SPSP queries may be answered using a number of different data structures. The GKT scheme makes use of the ***SP-matrix*** [32]. For a graph $G = (V, E)$, the SP-matrix M is a $|V| \times |V|$ matrix defined as follows. Entry $M[i, j]$ stores the second vertex along the shortest path from vertex v_i to v_j; if no such path exists, then it stores $\perp$. An SPSP query (v_i, v_j) is answered by computing $M[i, j] = v_k$ to obtain the next vertex along the path and then recursing on (v_k, v_j) until $\perp$ is returned.

At a high level, the GKT scheme proceeds by computing an SP-matrix for the query graph and then using this matrix to compute a dictionary SPDX'. This dictionary is then encrypted using a dictionary encryption scheme (DES) such as [23, 29]. To ensure that the GKT scheme is non-interactive, the underlying DES must be response-revealing. Since it is germane to our analysis, we provide the syntax of a DES next.

Definition 16. *A **dictionary encryption scheme (DES)** is a tuple of four algorithms* $\mathsf{DES} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Token}, \mathsf{Get})$ *with the following syntax:*

- $\mathsf{DES.KeyGen}$ *is probabilistic and takes as input security parameter* λ *and outputs key* k*.*
- $\mathsf{DES.Encrypt}$ *takes as input a secret key* k *and dictionary* D *and outputs an encrypted dictionary* ED*.*
- $\mathsf{DES.Token}$ *takes as input a key* k *and a label* lab *and outputs a search token* tk*.*
- $\mathsf{DES.Get}$ *takes as input a search token* tk *and an encrypted dictionary* ED *and returns a plaintext value* val*.*

Correctness for a DES DES states that for all dictionaries D, for all keys k output by $\mathsf{DES.KeyGen}$ and for pairs $(\mathsf{lab}, \mathsf{val})$ in D, executing $\mathsf{DES.Get}$ on input $\mathsf{tk} = \mathsf{DES.Token}(k, \mathsf{lab})$ and dictionary $\mathsf{ED} = \mathsf{DES.Encrypt}(k, \mathsf{D})$ results in output val.

Note that while the GKT scheme itself is response-hiding (i.e. the shortest path is not returned in plaintext to the client), the underlying DES used in the scheme is response-revealing, that is, the values in its encrypted dictionary ED are revealed at query time. The response-revealing property of the DES is necessary to enable the GKT scheme to operate in a non-interactive manner.

Now we provide a detailed description of the GKT scheme. At setup, the client generates two secret keys: one for a symmetric encryption scheme SKE, and one for a dictionary encryption scheme DES. It takes the input graph G and computes the SP-matrix $M[i, j]$. It then computes a dictionary SPDX such that for each pair of vertices $(v_i, v_j) \in V \times V$, we set $\mathsf{SPDX}[(v_i, v_j)] = (w, v_j)$ if $i \neq j$ and in the SP-matrix we have $M[i, j] = w$ for some vertex w.

The client then computes a second dictionary SPDX' as follows. For each label-value pair $(\mathsf{lab}, \mathsf{val})$ in SPDX the following steps are carried out. A search token tk is computed from val using algorithm $\mathsf{DES.Token}$ and a ciphertext c is computed by encrypting val using $\mathsf{SKE.Encrypt}$. Then $\mathsf{SPDX}'[\mathsf{lab}]$ is set to (tk, c). The resulting dictionary SPDX' is then encrypted using $\mathsf{DES.Encrypt}$ to produce an output EDB, which is given to the server.

Now the client can issue an SPSP query for a vertex pair (u, v) by generating a search token tk for (u, v) and sending it to the server. The server initializes an empty string resp and uses tk to search EDB and obtain a response a. If $a = \perp$, then it returns resp. Otherwise, it parses a as (tk', c), updates $\mathsf{resp} = \mathsf{resp}||c$ and recurses on tk' until $\perp$ is reached on look-up. The server returns resp, a concatenation of ciphertexts (or $\perp$) to the client. The client then uses its secret key to decrypt resp, obtaining a sequence of pairs $\mathsf{val} = (w_k, v)$ from which the shortest path from u to v can be constructed.

Complexity. The GKT scheme's setup takes time $O(n^3)$ and is dominated by the cost of computing the SP-matrix. Token generation takes time $O(1)$ (assuming use of an efficient DES) and querying EDB takes time $O(t)$ where t is the maximum length of a shortest path in G. The server storage is $O(n^2)$.

5.2.2 Leakage of the GKT Scheme

Ghosh et al. [52] provide a formal specification of their scheme's leakage. Informally, the setup leakage of their scheme is the number of vertex pairs in G that are connected by a path, while the query leakage consists of the query pattern (which pairs of queries are equal), the path intersection pattern (the overlap between pairs of shortest paths seen in queries), and the lengths of the shortest paths arising in queries. See [52, Section 4.1] for more details.

Recall that in the GKT scheme, the server obtains EDB by encrypting the underlying dictionary SPDX', in which labels are of the form $\mathsf{lab} = (v_i, v_j)$ and values are of the form $\mathsf{val} = (\mathsf{tk}, c)$, using a DES. Here tk is a search token obtained by running $\mathsf{DES.Token}$ on a pair (w, v_j) and c is obtained by running $\mathsf{SKE.Encrypt}$ also on (w, v_j). Since EDB is obtained by running DES on SPDX', this means that the labels in EDB are derived from tokens obtained by running $\mathsf{DES.Token}$ on inputs $\mathsf{lab} = (v_i, v_j)$. Moreover, these tokens also appear in the values in EDB that are revealed to the server at query time, that is, in the entries (tk, c).

In turn, the query leakage reveals to the server the token used to initiate a search, as well as all the subsequent pairs (tk, c) that are obtained by recursively processing such a query. Let us denote the sequence of search tokens associated with the processing of some (unknown) query q for a shortest path of length t as $s = \mathsf{tk}_1 \| \mathsf{tk}_2 \| \ldots \| \mathsf{tk}_{t+1} \in \{0,1\}^*$. We refer to this string as the ***token sequence of*** q. Since the search tokens correspond to the sequence of vertices in the queried path, there are as many tokens in the sequence as there are vertices in the shortest path for the query. Note that, by correctness of DES used in the construction of EDB, no two distinct queries can result in the same token sequence (in fact no two distinct queries can produce the same first token tk_1, since each such first token must be used to derive a unique label in EDB identifying the beginning of a specific shortest path).

Notice also that token sequences for different queries can be overlapping; indeed since the tokens are computed by running $\mathsf{DES.Token}$ on inputs $\mathsf{lab} = (v_i, v)$ where v is the final

vertex of a shortest path, two token sequences are overlapping if and only if they correspond to queries (and shortest paths) having the same end vertex. Hence, given the query leakage of a set of queries, the adversary can compute all the token sequences and construct from them $n' \leq n$ directed trees, $\{Q_i\}_{i\in[n']}$, each tree having at most n vertices and a single root vertex. The vertices across all n' trees are labelled with the search tokens in EDB and there is a directed edge from tk to tk' if and only if tk and tk' are adjacent in some token sequence. (Each tree has at most n vertices because of our assumption about G being connected.)

We call this set of trees the ***query trees***. Each query tree corresponds to the set of queries having the same end vertex. Each tree has a single sink (root) that corresponds to a unique vertex $v \in V$. The tree paths correspond to the shortest paths from vertices $w \in V \setminus \{v\}$ to v, such that w and v are connected in G. We note that Ghosh et al. [52] also discuss these trees but they do not analyze the theoretical limits of what can be inferred from them.

We denote the leakage of the GKT scheme on a graph G after issuing a set of SPSP queries $\mathcal{Q}$ as $\mathcal{L}(G, \mathcal{Q})$. For a formal proof of security that establishes the leakage profile of the GKT scheme please refer to [52]. We stress that our attacks are based only on the leakage of the scheme, as established above, and not on breaking the underlying cryptographic primitives of the scheme.

5.2.3 Implications of Leakage

Suppose that all queries have been issued and that we have constructed all n query trees $\{Q_i\}_{i\in[n]}$, each tree having n vertices. We observe that there exists a one-to-one matching between the query trees $\{Q_i\}_{i\in[n]}$ and the SDSP trees $\{T_v\}_{v\in V}$ of G such that each matched pair of trees is isomorphic. The reason is that the query trees are just differently labelled versions of the SDSP trees; in turn, this stems from the fact that paths in the query trees are in 1-1 correspondence with the shortest paths in G.

This now reveals the core of our query recovery attack, developed in detail in Section 5.3

below. The server with access to G first computes all the SDSP trees offline. As queries are issued, it then constructs the query trees one path at a time. Once a complete query tree Q is computed (recall that each query tree must have n vertices since G is connected) the server finds all possible isomorphisms between Q and the SDSP trees. Then, for each token sequence in Q, it computes the set of paths in the SDSP trees to which that token sequence can be mapped under the possible isomorphisms. This set of paths yields the set of possible queries to which the token sequence can correspond. This information is stored in a pair of dictionaries, which can be used to look up the candidate queries.

To illustrate the core attack idea, Figure 5.1 depicts (5.1a) a graph G, (5.1b) its SDSP tree for vertex 1 (with vertex labels and canonical names), and (5.1c) the matching query tree (without vertex labels). It is then clear that the leakage from the unique shortest path of length 2 in Figure (5.1c) can only be mapped to the corresponding path with edges $(4, 5)$, $(5, 1)$ in Figure (5.1b) under isomorphisms, and similarly the shortest path of length 1 that is a subpath of that path of length 2 can only be mapped to path $(5, 1)$. On the other hand, the 3 remaining paths of length 1 can be mapped under isomorphisms to ***any*** of the length 1 paths $(2, 1)$, $(3, 1)$, or $(6, 1)$ and so cannot be uniquely recovered.

Since the adversary only learns the query trees and token sequences from the leakage, the degree of query recovery that can be achieved based on that leakage is limited. In particular, without auxiliary information, the adversary can only recover the candidate queries up to symmetries arising from the isomorphisms between the query trees and the SDSP trees. In section 5.4, we show that in practice this is often not an issue since many queries result in only a very small number of candidate queries.

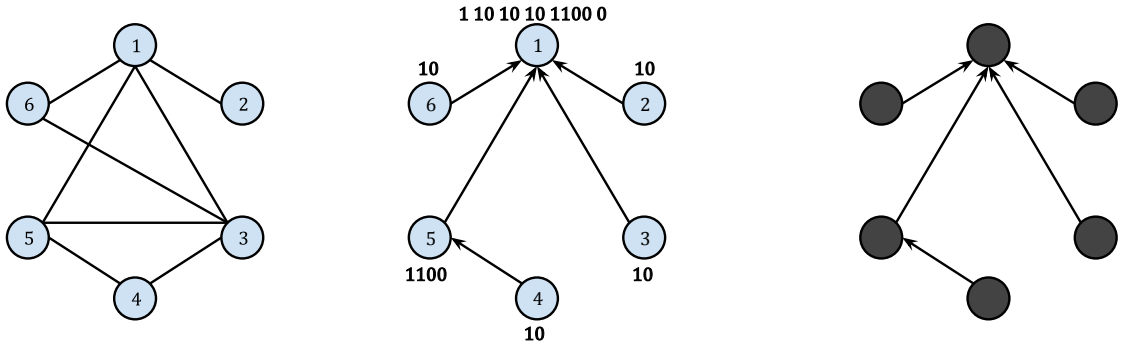
5.3 Query Recovery

5.3.1 Threat Model and Assumptions

We consider a ***passive, persistent, honest-but-curious*** adversary that has compromised the server and can observe the initial search token issued, all subsequent search tokens revealed during the query processing, and the response. In particular, this adversary could be the server itself. In Section 5.5 we briefly outline a modified version of our attack in which we assume that the adversary has only compromised the communication channels between the client and server, and can thus only see the search tokens used to initiate the recursive look-up and the server responses.

We assume that the adversary knows the graph G that has been encrypted to create EDB. As noted previously, this is a strong assumption, but fits within the security model used in [52] (where G can even be chosen) and is realistic in many routing/navigation scenarios. We further assume that the adversary sees enough queries to construct a subset of the n query trees. We emphasize that computing all n trees does ***not*** require observing all possible queries; in the real world datasets we tested, we were able to construct all query trees with as few as 68.1% of the possible queries. This is because constructing a query tree that corresponds to T_v only requires observing the queries that start at the leaf nodes of T_v and end at v. In SDSP trees with few leaves, only a small fraction of queries is needed.

We assume that the all-pairs shortest path algorithm used in constructing the SP-matrix from G during setup is deterministic. We assume that this algorithm is known to the adversary. Such an assumption is reasonable as the adversary knows G and many shortest path algorithms are deterministic, including Floyd-Warshall [47] and many of its adaptations.



(a) Original graph G. (b) SDSP tree for node 1. (c) The inferred leakage.

Figure 5.1: (a) Original graph G, (b) its corresponding SDSP tree for vertex 1 in G with the canonical names labeling all the vertices of the tree, and (c) the the matching query tree that is leaked during setup (without any vertex labels).

5.3.2 Formalising Query Recovery Attacks

Query recovery (QR) in general is the goal of determining the plaintext value of queries that have been issued by the client. The notion of query recovery was introduced by Islam et al. [64] in the context of leakage abuse attacks on SSE schemes and has been extensively studied in the context of SSE and related schemes since.

We study the problem of query recovery in the context of GESs, specifically, the GKT scheme: given G, the setup leakage of the GKT scheme and the query leakage from a set of SPSP queries, our adversary's goal is to match the leakage for each SPSP query with the corresponding start and end vertices (u, v) of a path in G. As noted above, there may be a number of candidate queries that can be assigned to the leakage from each query. We now formally describe the adversary's goals.

Definition 17. *(Consistency) Let $G = (V, E)$ be a graph, $\mathcal{Q} = \{q_1, \ldots, q_k\}$ be the set of SPSP queries that are issued, and $S = \{s_1, s_2, \ldots, s_k\}$ be the set of token sequences of the queries issued. An assignment $\pi : S \to V \times V$ is a mapping from token sequences to SPSP queries. An assignment π is said to be* ***consistent with the leakage*** *$\mathcal{L}(G, \mathcal{Q})$ if it satisfies $\mathcal{L}(G, \mathcal{Q}) = \mathcal{L}(G, \pi(S))$.*

Informally, consistency requires that, for each $s_i \in S$, the query $\pi(s_i)$ specified by assignment π ***could*** feasibly result in the observed leakage $\mathcal{L}(G, \mathcal{Q})$.

Definition 18. *(QR) Let $G = (V, E)$ be a graph, $\mathcal{Q} = \{q_1, \ldots, q_k\}$ be a set of SPSP queries, and S the corresponding set of token sequences. Let Π be the set of all assignments consistent with $\mathcal{L}(G, \mathcal{Q})$. The adversary achieves **query recovery (QR)** when it computes and outputs a mapping: $s \mapsto \{\pi(s) : \pi \in \Pi\}$ for all $s \in S$.*

Informally, the adversary achieves query recovery if, for each $s \in S$ (a set of token sequences resulting from queries in $\mathcal{Q}$), it outputs a set of query candidates $\{\pi(s) : \pi \in \Pi\}$ containing every query that is consistent with the leakage. Note that this implies that the output always contains the correct query (and possibly more). This is the best the adversary can do, given the available leakage.

There is some information not conveyed in this mapping. In particular, by fixing an assignment for a given token sequence, we may fix or reduce the possible assignments for other query responses. We give such an example below.

Example 5.3.1. *Suppose we observe the set of token sequences $\{s_i : i \in [5]\}$ such that s_1, s_2, s_3, s_4 correspond to paths of length 1 and s_5 corresponds to a path of length 2, with s_4 a subsequence of s_5, and which allows us to construct the query tree in Figure 5.1c. Further suppose that the resulting query tree is not isomorphic to any other query tree, so we know that all queries in S are rooted at 1. An adversary achieving QR must output the following:*

$$\{s_1 : \{(6,1),(3,1),(2,1)\}, s_2 : \{(6,1),(3,1),(2,1)\},$$
$$s_3 : \{(6,1),(3,1),(2,1)\}, s_4 : \{(5,1)\}, s_5 : \{(4,1)\}\}.$$

However, if the adversary could fix the assignment s_1 to $(1,6)$ (for example, by using auxiliary information) then s_2 could only be mapped to either $(1,3)$ or $(1,2)$.

We now define a special type of query recovery when there exists only one assignment consistent with the query leakage, i.e. all queries can be uniquely recovered.

Definition 19. *(FQR) Let $G = (V, E)$ be a graph, $\mathcal{Q} = \{q_1, \ldots, q_k\}$ be a set of SPSP queries, and S the corresponding set of token sequences. Let Π be the set of assignments consistent with $\mathcal{L}(G, \mathcal{Q})$. We say that the adversary achieves **full query recovery (FQR)** when it (a) achieves QR, and (b) $|\Pi| = 1$.*

That is, there is a unique assignment of token sequences to queries consistent with the leakage. Whether FQR is ***always possible*** (i.e. for every possible set of queries $\mathcal{Q}$) depends on the graph G. Specifically, we will see that FQR is always possible if and only if each SDSP tree arising in G is non-isomorphic and every path in each SDSP tree is fixed by all automorphisms of the tree. It is easy to construct graphs for which these conditions hold (see Section 5.3.10). For such graphs, our QR attack always achieves FQR.

5.3.3 Technical Results

We develop some technical results concerning isomorphisms of trees and the behavior of paths under those isomorphisms that we will need in the remainder of the chapter.

For any rooted tree $T = (V, E, r)$ and any $u \in V$, let $T[u] \subseteq T$ denote the subtree induced by u and all its descendants in T.

Lemma 5.3.2. *Let $T = (V, E, r)$ and $T' = (V', E', r')$ be rooted trees, and $p_{u,r} = (u, w_1, \ldots, w_t, r)$ and $p_{v,r'} = (v, w'_1, \ldots, w'_\ell, r')$ be paths in T and T', respectively. If there exists an isomorphism $\varphi : T \to T'$ such that $\varphi(u) = v$, then $t = \ell$ and $\varphi(w_i) = w'_i$ $\forall i \in [t]$.*

Proof. By assumption $\varphi(u) = v$ and by definition of isomorphism of rooted trees we also have that $\varphi(r) = r'$. Since T is a tree, then we know that there exists a unique path between u and r, and between v and r'. Isomorphisms of graphs must be edge preserving, and so φ

must map the subgraph $p_{u,r}$ to $p_{v,r'}$. These two paths can only be isomorphic if they are the same length and thus $t = \ell$. Putting together these two facts we have that

$$(\varphi(u), \varphi(w_1)) = (v, w_1'), (\varphi(w_1), \varphi(w_2)) = (w_1', w_2'), \ldots, (\varphi(w_t), \varphi(r)) = (w_t, r')$$

which concludes the proof. □

Given a rooted tree $T = (V, E, r)$ and $u \in V$, let $\mathsf{PathName}_T(u)$ denote the concatenation of the canonical names of vertices along the path from u to r in T, separated by semicolons:

$$\mathsf{PathName}_T(u) = \mathsf{Name}(T[u]) \| \text{“;”} \| \mathsf{Name}(T[w_1]) \| \text{“;”} \| \ldots \| \text{“;”} \| \mathsf{Name}(T[w_t]) \| \text{“;”} \| \mathsf{Name}(T[r]).$$

Computing ***path names*** will form the core of our QR attack. Before we explain how we use them, we prove a sequence of results about the relationship between path names and isomorphisms. In Section 5.3.5 we explain how to apply a universal hash function to the path names to compress their length from $O(n^2)$ to $O(\log n)$ bits, thereby reducing storage and run time complexity.

Proposition 5.3.3. *Let $T = (V, E, r)$ and $T' = (V', E', r')$ be isomorphic rooted trees and let C and C' denote the set of children of r and r', respectively. There is an isomorphism from T to T' if and only if there is a perfect matching from C to C' such that for each matched pair $c_i \in C, c_i' \in C'$, there exists an isomorphism $\varphi_i : T[c_i] \to T[c_i']$.*

Proof. To see the forwards direction, let φ denote an isomorphism from T to T' and note that if $\varphi(c) = c'$ for $c \in C$, then by the edge-preservation property of isomorphisms, φ must map the vertices of $T[c]$ to the vertices of $T[c']$, and thus $T[c] \cong T[c']$. For the backwards direction, we construct an isomorphism φ from T to T'. Let φ_r be the trivial isomorphism that takes r to r' and let

$$\varphi = \varphi_1 \cup \varphi_2 \cup \cdots \cup \varphi_k \cup \varphi_r.$$

Let $(a, b) \in E$. If (a, b) is an edge in $T[c_i]$ for some $c_i \in C$ then it is easy to see that by restricting φ to the vertices in $T[c_i]$, we have that $(\varphi(a), \varphi(b))$ is an edge in $T'[c'_i] \subseteq T'$. If $(a, b) = (c_i, r)$ for some $c_i \in C$, then $(\varphi(c_i), \varphi(r)) = (c'_i, r')$. Since c'_i is a child of r' then $(\varphi(a), \varphi(b)) \in E'$. A similar argument holds for showing that if $(a, b) \in E'$, then $(\varphi^{-1}(a), \varphi^{-1}(b)) \in E$. □

Lemma 5.3.4. *Let $T = (V, E, r)$ and $T' = (V', E', r')$ be isomorphic rooted trees. Let u and v be children of r and r', respectively. Suppose that σ is an isomorphism from $T[u]$ to $T'[v]$. Then there exists an isomorphism φ from T to T' such that $\varphi|_{T[u]} = \sigma$ and $\varphi(u) = v$.*

Proof. Let C and C' denote the set of children of r and r', respectively. Since φ is an isomorphism and is edge preserving, then it must map C to C' and we necessarily have that $k = |C| = |C'|$.

We now use Proposition 5.3.3 to prove the lemma. Let $\hat{\varphi}$ be any isomorphism from T to T'. If $\hat{\varphi}$ maps u to v then we are done. Otherwise, $\hat{\varphi}(u) = c' \neq v$ and $\hat{\varphi}^{-1}(v) = c \neq u$ for some $c' \in C'$ and $c \in C$. By Proposition 5.3.3 we have that $T[u] \cong T'[c']$ and $T[c] \cong T'[v]$, and by assumption we also know that $T[u] \cong T'[v]$. Thus, by transitivity, we conclude that $T[c] \cong T'[c']$. Let W be the vertices in $T \setminus (T[u] \cup T[c])$ and let π be an isomorphism from $T[c]$ to $T'[c']$. Then $\varphi = \hat{\varphi}|_W \cup \sigma \cup \pi$ is a collection of isomorphisms on all the trees rooted at the children of the roots. Thus φ is an isomorphism from T to T' that maps u to v. □

We now come to our main technical result:

Theorem 5.3.5. *Let $T = (V, E, r)$ and $T' = (V', E', r')$ be rooted trees and let $u \in V$ and $v \in V'$. There exists an isomorphism $\varphi : T \to T'$ mapping u to v if and only if* $\mathsf{PathName}_T(u) = \mathsf{PathName}_{T'}(v)$.

Proof. The forwards direction follows from Lemma 5.3.2.

For the backwards direction, suppose that $\mathsf{PathName}_T(u) = \mathsf{PathName}_{T'}(v)$. Since a path name includes the canonical name of the entire tree, we deduce that $\mathsf{Name}(T[r]) = \mathsf{Name}(T'[r'])$; it follows that $T \cong T'$. Similarly we deduce that $T[u] \cong T'[v]$. More generally, let $p_{u,r} = (u, w_1, \ldots, w_{t-1}, r)$ and $p_{v,r'} = (v, w'_1, \ldots, w'_{t-1}, r')$ be paths in T and T', respectively. Then for all $i \in [t-1]$ we have that $\mathsf{Name}(T[w_i]) = \mathsf{Name}(T'[w'_i])$.

We now prove the result inductively on the vertices along the path from u to r. For the base case, take any isomorphism φ_0 from $T[u]$ to $T'[v]$ and note that this must necessarily map u to v. We can now extend this reasoning level-by-level upwards, at each stage using the equalities of components of the two path names to extend the isomorphism. Suppose that for $k \leq t-1$ there exists an isomorphism φ_k from $T[w_k]$ to $T'[w'_k]$ such that $\varphi_k(u) = v$. By equality of path names we have that $T[w_{k+1}] \cong T'[w'_{k+1}]$. Note also that w_k and w'_k are children of w_{k+1} and w'_{k+1}, respectively. Applying Lemma 5.3.4, we see that there exists an isomorphism φ_{k+1} from $T[w_{k+1}]$ to $T'[w'_{k+1}]$ such that $\varphi_{k+1}|_{T[w_k]} = \varphi_k$. Since u is a vertex in $T[w_k]$, it follows that $\varphi_{k+1}(u) = \varphi_k(u) = v$. This completes the proof. □

Theorem 5.3.5 also gives us a method for identifying when there exists only a single isomorphism between two rooted trees. Suppose that $T = (V, E, r)$ and $T' = (V', E', r')$ are isomorphic rooted trees and that every vertex $v \in V$ has a distinct path name; then there exists exactly one isomorphism from T to T'. Intuitively, a vertex in T can only be mapped to a vertex in T' with the same path name. So if path names are unique, then each vertex in T can only be mapped to a single vertex in T', meaning there is only a single isomorphism available. The converse also holds: if there exists exactly one isomorphism from T to T', then every vertex $v \in V$ necessarily has a distinct path name. This observation will be useful in characterizing when query reconstruction results in full query recovery. We summarise with:

Corollary 5.3.6. *Let T and T be isomorphic rooted trees. Every vertex v in T has a unique path name in T if and only if there exists a single isomorphism from T to T'.*

5.3.4 Overview of the Query Recovery Attack

Our QR attack takes as input the graph G, a set of token sequences corresponding to the set of issued queries, and comprises of the following steps:

0. **Preprocess the graph offline** (Algorithm 18). Compute the SDSP trees $\{T_v\}_{v \in V}$ of graph G. Then compute a multimap M that maps each path name arising in the T_v to the set of SPSP queries whose start vertices have the same path name.

1. **Compute the query trees online**. The trees are constructed from the token sequences as the queries are issued.

2. **Process the query trees** (Algorithm 19). Compute a dictionary D that maps each token sequence to the path name of the start vertex of the path.

Note that steps 0 and 2 are trivially parallelizable. In the case that the APSP algorithm is randomized, the adversary can simply run the attack multiple times to account for different shortest path trees.

In practice, the attack can output a single large table T matching token sequences s to sets of queries. However, storing this large table will be more expensive than storing D and M when G has high symmetry. Moreover, D can be indexed by the first token tk in each token sequence s (since tk uniquely determines the sequence).

In the following subsections, we expand on each of the steps in the above overview.

5.3.5 Computing the Path Names

Before diving into our attack, we describe our algorithm for computing path names which we use as a subroutine of our attack. Algorithm 17 (ComputePathNames) takes as input a rooted tree $T = (V, E, r)$ and outputs a dictionary mapping each vertex $v \in V$ to its path name. First, we call Algorithm 1 (ComputeNames) on tree T, its root r, and an

empty dictionary Names, and obtain a dictionary Names that maps each vertex $v \in V$ to the canonical name of subtree $T[v]$.

We first recall the definition of a universal hash function. A set H of functions $U \to [M]$ is a ***universal hash function family*** if, for every distinct $x, y \in U$ the hash function family H satisfies the following constraint:

$$\Pr_{h \leftarrow H}[h(x) = h(y)] \leq 1/M.$$

We will use a function h drawn from a universal hash function family H to compress the path names from $O(n^2)$ to $O(\log n)$. We initialize an empty dictionary PathNames and set $\mathsf{PathNames}[r] = h(\mathsf{Names}[r])$. We then traverse T in a depth first search manner; when a new vertex v is discovered during the traversal, we set $\mathsf{PathNames}[v]$ to the hash of the concatenation of the name of v and the path name of its parent u i.e.

$$\mathsf{PathNames}[v] = h(\mathsf{Names}[v] \| \mathsf{PathNames}[u]). \tag{5.1}$$

When all vertices have been explored, PathNames is returned. The pseudocode can be found in Algorithm 17.

Theorem 5.3.7. *Let T be a rooted tree and* PathNames *be the output of running Algorithm 17 on T. Let H be a universal hash function family mapping $\{0,1\}^* \to \{0,1\}^{6 \log n}$. Then for randomly sampled $h \leftarrow H$ the expected number of collisions in* PathNames *is at most $O(1/n^3)$.*

Proof. Let $u, v \in V$ be distinct and let $(u, w_1, \ldots, w_k, r = w_{k+1})$ and $(v, w'_1, \ldots, w'_{k'}, r = w_{k'+1})$ be paths in T. We thus have

$$\mathsf{PathNames}[u] = h(\mathsf{Names}[u] \| h(\mathsf{Names}[w_1] \| h(\mathsf{Names}[w_2] \| ...)))$$

and similarly for $\mathsf{PathNames}[v]$. For there to be a collision between their path names

then either: (1) $\mathsf{Names}[u]\|\mathsf{PathNames}[w_1]$ and $\mathsf{Names}[v]\|\mathsf{PathNames}[w'_1]$ collide or (2) for some $i \in [\min\{k, k'\}]$ and $k, k' \leq n-2$, we have that $\mathsf{Names}[w_i]\|\mathsf{PathNames}[w_{i+1}]$ and $\mathsf{Names}[w'_i]\|\mathsf{PathNames}[w'_{i+1}]$ collide. Recall that a canonical name is unique up to isomorphism of the rooted tree.

Let C_{uv} denote the event that the path names of u and v collide, and let C^j_{uv} denote the event that the j-th nested hash of u's and v's path names collide. A collision on the path names occurs when any of the at most n pairs of nested hash values (used to compute the path names of u and v) collide. By definition of universal hash function we have $\mathbb{E}[C^j_{uv}] < 1/n^6$. Thus, by linearity of expectation,

$$\mathbb{E}[C_{uv}] = \sum_{j=1}^{\min\{k+1,k'+1\}} \mathbb{E}[C^j_{uv}] < \frac{n}{n^6} = \frac{1}{n^5}.$$

Let C denote the event of any collision of path names in T. Then by linearity of expectation the expected number of collisions is

$$\mathbb{E}[C] = \sum_u \sum_v \mathbb{E}[C_{uv}] < \frac{n^2}{n^5} = \frac{1}{n^3}.$$

□

Corollary 5.3.8. *Let $G = (V, E)$ be a graph and let $\{T_r\}_{r \in V}$ be the set of SDSP trees of G. Let* PathNames *be the union of the outputs of running Algorithm 17 on each tree in $\{T_r\}_{r \in V}$. Let H be a universal hash function family mapping $\{0,1\}^* \to \{0,1\}^{6 \log n}$. Then for randomly sampled $h \leftarrow H$ the expected number of collisions in* PathNames *is at most $O(1/n)$.*

We note that to achieve a smaller probability of collision, one can choose a hash function family H whose output length is $c \log n$ where $c > 6$. For simplicity we invoke the universal hash function using SHA-256 truncated to 128 bits.

Algorithm 17 ComputePathNames

Input: Rooted tree $T = (V, E, r)$.
Output: Dictionary PathNames.

```
1:  // Compute the canonical name of T[v] for all v ∈ V.
2:  Initialize empty dictionaries Names and PathNames
3:  Initialize empty stack S
4:  Names ← ComputeNames(T, r, Names)
5:  h ← H
6:
7:  // Concatenate the canonical names into path names.
8:  S.push(r)
9:  Mark r as explored
10: PathNames[r] ← h(Names[r])
11:
12: while S ≠ ∅ do
13:     v ← S.pop( )
14:     if v is not explored then
15:         Let u be the parent of v
16:         PathNames[v] = h(Names[v] ‖ ";" ‖ PathNames[u])
17:         Mark v as explored
18:     for children w of v do
19:         S.push(w)
20: return PathNames
```

Lemma 5.3.9. *Let $T = (V, E, r)$ be a rooted tree on n vertices and H be a universal hash function family mapping $\{0,1\}^* \to \{0,1\}^{6 \log n}$. Upon input of T, Algorithm 17 returns a dictionary of size $O(n \log n)$ mapping each $v \in V$ to a hash of its path name in time $O(n^2)$.*

Proof. Correctness follows easily from Theorem 5.3.7 and by a recursive argument.

Calling ComputeNames (Algorithm 1) takes $O(n^2)$ time. Reading the name of the root r and assigning the hash of its name takes at most time $O(n)$. Every node is pushed onto the stack once, and thus the **while** loop on line 12 iterates n times. Assigning a new path name on line 16 takes time $O(n)$ since Names$[v]$ is $O(n)$ bits, PathNames$[u]$ is $O(\log n)$ bits, and computing the hash takes constant time. Pushing the children of a given vertex onto the stack takes time $O(n)$ for a total run time of $O(n^2)$. PathNames maps the vertices to the hash of their path names. Each vertex and its hashed path name can be encoded with $O(\log n)$ bits yielding a dictionary of size $O(n \log n)$. □

5.3.6 Preprocess the Graph

We first preprocess the original graph $G = (V, E)$ into the n SDSP trees. Since the adversary is assumed to have knowledge of G, this step can be done offline. We use the same all-pairs shortest paths algorithm used at setup on G to compute the n SDSP trees $\{T_v\}_{v \in V}$, where tree T_v is rooted at vertex v. For unweighted, undirected graphs, we can use breadth first search for a total run time of $O(n^2 + nm)$ where $m = |E|$; For general weighted graphs this has step has a run time of $O(n^3)$ [47].

Next, we compute the path names of each vertex in $\{T_r\}_{r \in V}$, and then construct a multimap M that maps the (hashed) path name of each vertex in $\{T_r\}_{r \in V}$ to the set of SPSP queries whose start vertices have the same path name. We leverage Theorem 5.3.5 to construct this map and describe the steps in detail below.

We initialize an empty multimap M. For each $r \in V$ we compute $\mathsf{PathNames}$ by running Algorithm 17 (ComputePathNames) on T_r. For each vertex v in T_r we compute $path\_name \leftarrow \mathsf{PathNames}[v]$, and check whether $path\_name$ is a label in M. If yes, $\mathsf{M}[path\_name] \leftarrow \mathsf{M}[path\_name] \cup \{(v, r)\}$. Otherwise $\mathsf{M}[path\_name] \leftarrow \{(v, r)\}$. The pseudocode for computing M can be found in Algorithm 18.

Lemma 5.3.10. *Let $G = (V, E)$ be a graph on n vertices. Upon input of G, Algorithm 18 returns a multimap of size $O(n^2 \log n)$ mapping each $v \in V$ to its corresponding path name in time $O(n^3)$.*

Proof. For each vertex $r \in V$ we compute a dictionary mapping each vertex in T_r to its respective path names. The correctness of path names follows from Lemma 5.3.9.

We now analyze the run time. Computing all-pairs shortest path takes time $O(n^3)$. The **for** loop on line 5 iterates through n vertices. For each vertex in V, we run Algorithm 17 (ComputePathNames) which takes $O(n^2)$ time and the inner **for** loop on line 9 takes $O(n)$ time. Thus, the **for** loop on line 5 takes a total time of $O(n^3)$.

Algorithm 18 PREPROCESSGRAPH

Input: A graph G.
Output: A multimap M mapping path names to sets of SPSP queries.

// Compute the set of SDSP trees from G.
Initialize an empty multimap M
Compute $\{T_v\}_{v \in V}$ by running all-pairs shortest path on G

for $r \in V$ **do**
    // Compute the path names of each vertex.
    $\mathsf{PathNames} \leftarrow$ COMPUTEPATHNAMES(T_r)
    // Map path names to candidate queries.
    for $(v, path\_name)$ in $\mathsf{PathNames}$ **do**
        if $path\_name$ is a label in M **then**
            $\mathsf{M}[path\_name] \leftarrow \mathsf{M}[path\_name] \cup \{(v, r)\}$
        else
            $\mathsf{M}[path\_name] \leftarrow \{(v, r)\}$
return M

The multimap maps hashes of the path names to a list of candidate queries. The hashed path names have size $O(\log n)$ and there are at most n^2 distinct path names; each query corresponds to only one path name and is $O(\log n)$ bits long. The multimap thus has total size $O(n^2 \log n)$. □

5.3.7 Process the Search Tokens

We must now process the tokens revealed at query time. Recall that the tokens are revealed such that, the response to any shortest path query can be computed non-interactively. When a search token tk is sent to the server, the server recursively looks up each of the encrypted vertices along the path. The adversary can thus compute the query trees using the search tokens revealed at query time. First, it initializes an empty graph F.

As label-value pairs $(\mathsf{lab}, \mathsf{val})$ are revealed in EDB, the adversary parses $\mathsf{tk}_{\mathsf{curr}} \leftarrow \mathsf{lab}$ and $(\mathsf{tk}_{\mathsf{next}}, c) \leftarrow \mathsf{val}$, and adds $(\mathsf{tk}_{\mathsf{curr}}, \mathsf{tk}_{\mathsf{next}})$ as a directed edge to F. At any given time, F will be a forest comprised of $n' \leq n$ trees, $\{Q_i\}_{i \in [n']}$, such that each Q_i has at most n nodes. Identifying the individual trees in the forest can be done in time $O(n^2)$. ***The adversary can compute the query trees online and the final step of the attack***

can be run on any set of complete query trees. A complete query tree corresponds to the set of all queries to some fixed destination vertex. For ease of explanation, we assume Algorithm 19 (QueryMapping) takes as input the set of all complete query trees that have been constructed from the leakage.

5.3.8 Map the Token Sequences to SPSP Queries

In the last step, we take as input the set of complete query trees $\{Q_i\}_{i\in[n']}$ constructed from the leakage. We use the path names of each vertex in the $\{Q_i\}_{i\in[n']}$, to construct a dictionary D that maps each token sequence s to the path name of the starting vertex of the corresponding path in its resspective query tree .

We first initialize an empty dictionary D. For each complete query tree Q_i, we compute $\mathsf{PathNames} \leftarrow \textsc{ComputePathNames}(Q_i)$ and take the union of $\mathsf{PathNames}$ and D. The pseudocode for computing D can be found in Algorithm 19.

Theorem 5.3.11. *Let $G = (V, E)$ be a graph and EDB be an encryption of G using the GKT scheme. Let $\{Q_i\}_{i\in[n']}$ be the query trees constructed from the leakage of queries issued to EDB. Upon input of G, Algorithm 18 returns a dictionary M mapping each path name to a set of SPSP queries in time $O(n^3)$. Upon input of G and $\{Q_i\}_{i\in[n']}$, Algorithm 19 returns a dictionary D mapping token sequences to path names in time $O(n^3)$. Moreover, the outputs D and M have the property that, for any token sequence s corresponding to a path (v, r) in a query tree and for every query $(v', r') \in \mathsf{M}[\mathsf{D}[s]]$, there exists an isomorphism φ from Q to $T_{r'}$ such that $\varphi(v) = v'$ and $\varphi(r) = r'$.*

Proof. The correctness of $\{Q_i\}_{i\in[n]}$ follows from the correctness of the GKT scheme. Dictionary D contains a map of each vertex in $\cup_{i\in[n]}Q_i$ to its path name. The correctness of M and D follows from Lemmas 5.3.10 and 5.3.9, respectively.

Let (v, r) be a pair comprised of a non-root vertex v and a root vertex r in a complete query tree Q, and let s be the token sequence corresponding to (v, r). Let $(v', r') \in \mathsf{M}[\mathsf{D}[s]]$.

Algorithm 19 QUERYMAPPING

Input: A graph G and a set of query trees $\{Q_i\}_{i\in[n']}$ with $n' \leq n$.
Output: A dictionary D mapping search tokens to path names, and a multimap M mapping path names to sets of SPSP queries.

```
Initialize empty dictionary D
for i ∈ [n'] do
    // Compute the path names of each vertex in the query trees.
    PathNames ← ComputePathNames(Q_i)
    D ← D ∪ PathNames
return D
```

By composition of D and M we must have that $\mathsf{PathName}_Q(v) = \mathsf{PathName}_{T_{r'}}(v')$. Applying Theorem 5.3.5, there is thus an isomorphism from Q to $T_{r'}$ that maps v to v' and r to r'.

We now analyze the run time. Preprocessing G (Algorithm 18) takes time $O(n^3)$. Computing the path names (Algorithm 17) of $n' \leq n$ trees takes $O(n^3)$ time, which gives us an upper bound on the run time of the whole attack. □

5.3.9 Recover the Queries

Once the map between each node (token) in a query tree and its corresponding path name has been computed, the attacker can use M and D to compute the candidate queries of all queries in the complete query trees. Given M and D (outputs of Algorithms 18 and 19, respectively) and an observed token s matching a query in the query trees for some unknown query, the adversary can find the set of queries consistent with s by simply computing $\mathsf{M}[\mathsf{D}[s]]$.

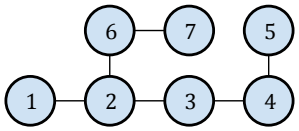


Figure 5.2: A graph for which FQR is always possible, no matter what set of queries is issued.

5.3.10 Full Query Recovery

We conclude this section with a discussion of when FQR is possible. By the correctness of our attack, this is the case for a graph G, a set of complete query trees $\{Q_i\}_{i\in[n']}$, and associated token sequences S when for $\mathsf{M} \leftarrow \textsc{PreprocessGraph}(G)$, $\mathsf{D} \leftarrow \textsc{QueryMapping}(G, \{Q_i\}_{i\in n'})$ and all $s \in S$ we have $|\mathsf{M}[\mathsf{D}[s]]| = 1$.

We can also phrase a condition for FQR feasibility in graph-theoretic terms. Recall Corollary 5.3.6, which states that given two isomorphic rooted trees T and T', if each vertex in T has a unique path name, then there exists only one isomorphism from T to T'. We deduce that FQR is always achievable for any set of complete query trees, when all n^2 vertices in the SDSP trees have unique path names. Formally, we have the following:

Corollary 5.3.12. *Let $G = (V, E)$ be a graph and let $\{T_v\}_{v\in V}$ be the set of SDSP trees of G. Suppose every vertex in $\bigcup_{v\in V} T_v$ has a unique path name (and in particular, each $T \in \{T_v\}_{v\in V}$ has a unique canonical name). Then FQR can always be achieved on any complete query tree(s). The converse is also true.*

By correctness, our attack achieves FQR whenever it is possible. In Figure 5.2, we show an example graph G for which FQR is always possible. Indeed, each tree $\{T_v\}_{v\in[7]}$ has a unique canonical name, and for all $v \in [7]$, each vertex u in T_v has a unique path name. More generally, let $\mathcal{G}$ be the family of graphs having one central vertex c and any number of paths all of distinct lengths appended to c. It is easy to see that our attack achieves FQR for all graphs $G \in \mathcal{G}$.

Dataset	n	m	d	# Comp	$\frac{\#\text{ Unique}}{\text{Total}}$	% Unique	$\frac{\#\text{ Leaves in SDSP trees}}{\#\text{ Nodes in SDSP trees}}$	% Min	Percentile 50	90	99
InternetRouting	35	323	0.543	1	$\frac{28}{1190}$	2.353 %	$\frac{1120}{1190}$	94.1 %	40	84	90
CA-GrQc	46	1030	0.995	1	$\frac{3}{2070}$	0.145 %	$\frac{2065}{2070}$	99.8 %	1845	1845	1845
email-Eu-core	1005	16,706	0.0331	20	$\frac{65,659}{1,009,020}$	6.507 %	$\frac{787,486}{1,009,020}$	78.0 %	16	69	190
facebook-combined	4039	88,234	0.011	1	$\frac{33,634}{16,309,482}$	0.206 %	$\frac{16,194,084}{16,309,482}$	99.3 %	1826	11,424	20,480
p2p-Gnutella08	6301	20,777	0.001	2	$\frac{8,519,868}{39,696,300}$	21.463 %	$\frac{27,663,800}{39,696,300}$	69.7 %	4	12	64
p2p-Gnutella04	10,876	39,994	0.0006	1	$\frac{25,915,785}{118,276,500}$	21.911 %	$\frac{80,580,827}{118,276,500}$	68.1 %	3	9	32
p2p-Gnutella25	22687	54705	0.0002	13	$\frac{82,736,533}{514,677,282}$	16.075 %	$\frac{379,383,168}{514,677,282}$	73.7 %	5	18	54
p2p-Gnutella30	36682	88328	0.0001	12	$\frac{197,413,906}{1,345,532,442}$	14.671 %	$\frac{1,003,317,663}{1,345,532,442}$	74.6 %	5	24	60

Table 5.1: A list of all real-world datasets used in our experiments; n denotes the number of vertices; m denotes the number of edges of the graph dataset; $d = 2m/(n \cdot (n-1))$ denotes the density of the graph. The last three columns show the 50th, 90th, and 99th percentiles obtained for Query Recovery on the eight real-world datasets.

5.4 Experimental Evaluation

5.4.1 Implementation Details

We implemented our attacks in Python 3.7.6 and ran our experiments on a computing cluster with a 2 x 28 Core Intel Xeon Gold 6258R 2.7GHz Processor (Turbo up to 4GHz / AVX512 Support), and 384GB DDR4 2933MHz ECC Memory. To generate the leakage, we implemented the GES from [52] and we used the same machine for the client and the server. The cryptographic primitives were implemented using the PyCryptodome library version 3.10.1 [107]; for symmetric encryption we used AES-CBC with a 16B key and for collision resistant hash functions we used SHA-256. For the DES, we implemeted Π_{bas} from [23] and generated the tokens using HMAC with SHA-256 truncated to 128 bits. The shortest paths of the graphs were computed using the `single_source_shortest_path` algorithm from the NetworkX library version 2.6.2 [98]. For our attacks, we used used the same shortest path algorithm from NetworkX as in our scheme implementation. We also used our own implementation of the AHU algorithm (Algorithm 1) to compute canonical names. The attack is highly parallelizable, and we exploited this property when implementing our attack.

5.4.2 Graph Datasets

We evaluate our attacks on 6 of the same data sets as [52]; in addition we use the InternetRouting dataset from the University of Oregon Route Views Project collected on January 2, 2000 and the facebook-combined dataset. All 8 of these datasets were obtained from [77]. The InternetRouting and CA-GrQc datasets were extracted from the original datasets using the dense subset extraction algorithm by Charikar [28] as implemented by Ambavi et al. [6]. Details about these datasets can be found in Table 5.1.

In addition to real world datasets, we deployed our attacks on random graphs for $n = 100, 250, 500, 1000$ and edge probabilities $p = 0.2, 0.4, 0.6, 0.8$. The graphs were generated using the `fast_gnp_random_graph` function from NetworkX [98].

5.4.3 Query Reconstruction Results

Real world datasets. We carried out our attack on the Internet Routing, CA-GrQc, email-EU-Core, facebook-combined, and p2p-Gnutella08 datasets; The online portion of the attack (Algorithm 19) given all queries ran in 0.087s, 0.093s, 5.807s, 102.670s, and 339.957s for each dataset, respectively. For the first four datasets, we also ran attacks given 75% and 90% of the queries averaged over 10 runs and sampled as follows: The start vertex was chosen uniformly at random and the end vertex was chosen with probability linearly proportional to its out degree in the original graph. This simulates a more realistic setting in which certain "highly connected" destinations are chosen with higher frequency. The results of these experiments can be found in Table 5.2. Queries can be reconstructed with just 75% of the queries. In fact, with high probability, we start seeing complete query trees with as few as 20% of the queries for the facebook-combined dataset.

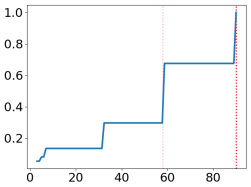
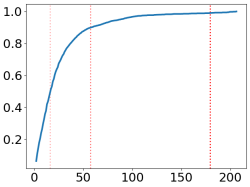
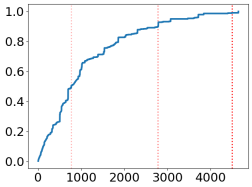
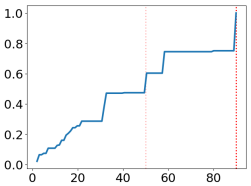
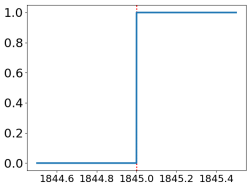
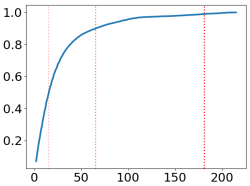
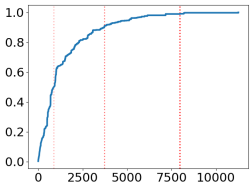
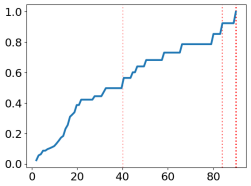
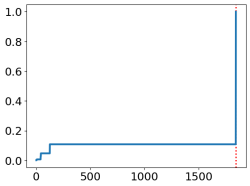
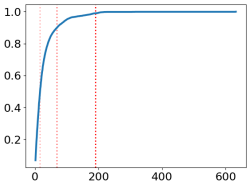
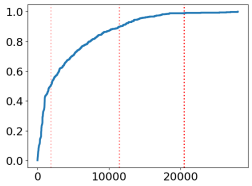
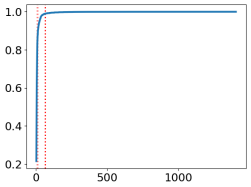
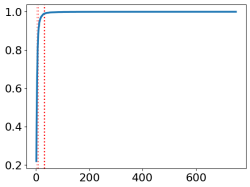
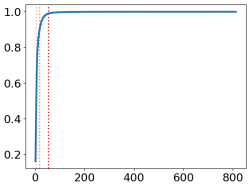
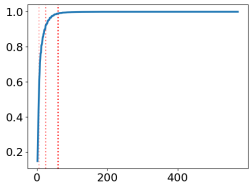
InternetRouting	Ca-GrQc	email-EU-core	facebook-combined
75% of the Queries (averaged over 10 runs)			
			
90% of the Queries (averaged over 10 runs)			
			
100% of the Queries			
			
p2p-Gnutella08	**p2p-Gnutella04**	**p2p-Gnutella25**	**p2p-Gnutella30**
100% of the Queries			
			

Table 5.2: CDFs for QR of the real-world data sets after observing (row 1) 75%, (row 2) 90%, and (rows 3 and 4) 100% of the queries. On the x axis we plot the number of candidate queries output by our attack and on the y axis we plot the percent of total queries. The red dotted lines indicate the 50th, 90th, and 99th percentiles. Because of Ca-GrQC's high symmetry, complete query trees could only be constructed after at least 80% of the queries were observed and hence its first graph is omitted.

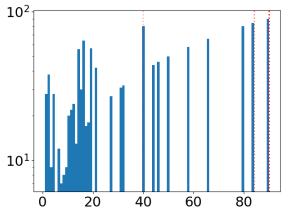
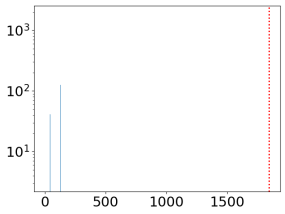
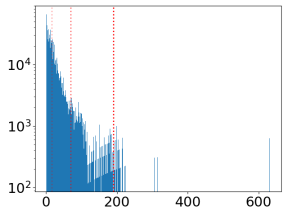
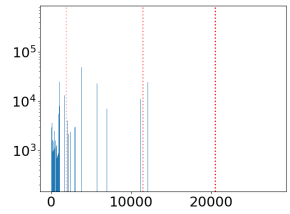
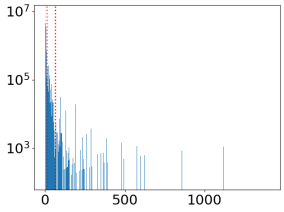
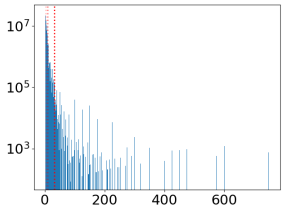
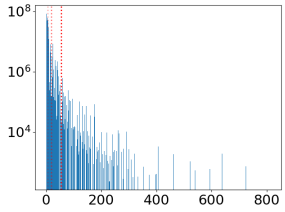
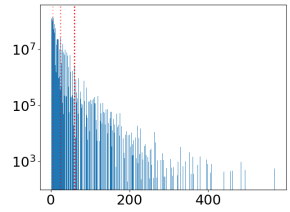
InternetRouting	Ca-GrQc	email-EU-core	facebook-combined
			
p2p-Gnutella08	**p2p-Gnutella04***	**p2p-Gnutella25***	**p2p-Gnutella30***
			

Table 5.3: Histograms for QR of the real world data sets after observing 100% of the queries. On the x axis we plot the number of candidate queries output by our QR attack and on the y axis we plot the number of queries. The red dotted lines indicate the 50th, 90th, and 99th percentiles. An asterisk next to the data set indicates that results were obtained via simulation, see discussion for details.

For the remaining datasets we ran simulations to demonstrate the success that an adversary could achieve given 100% of the queries. Our simulations were carried out as follows. Given G, the SDSP trees and the path names for each vertex in these trees were computed, and then a dictionary mapping each query in G to the set of candidate queries was constructed by identifying queries whose starting vertices have the same path name. The simulations only used the plaintext graph and the results show the success that an adversary would achieve in an end-to-end attack. These results can be found in the bottom row of Table 5.2.

In Table 5.1 we report the percent of uniquely recoverable queries when the attack is run on the set of all query trees. ***Uniquely recoverable queries*** are queries whose responses result in only one candidate. CA-GrQc had the smallest percentage of uniquely recoverable queries (0.145%) and the p2p-Gnutella04 had the largest percentage (21.911%). The small

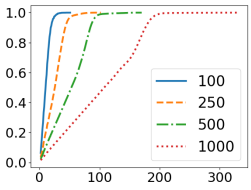
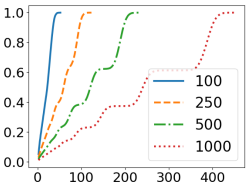
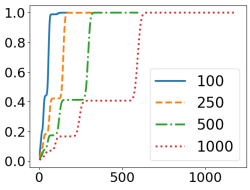
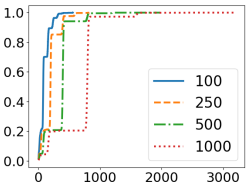
$p = 0.2$	$p = 0.4$	$p = 0.6$	$p = 0.8$
			

Table 5.4: CDFs for QR of random graphs for $n = 100, 250, 500, 1000$ and $p = 0.2, 0.4, 0.6, 0.8$ after observing 100% of the queries. On the x axis we plot the number of candidate queries output by our QR attack and on the y axis we plot the percent of total queries. For each (n, p) we generated 50 graphs and took an average of the number of vertices with each given set size of candidate queries. We observe that as the edge probability increases, the number of symmetries, and hence the number of candidate queries output tends to increase.

percentage for CA-GrQc can be attributed to its high density ($d = 0.995$), where density is defined as $d = 2m/(n \cdot (n-1))$. The CA-GrQc graph is nearly complete, and its SDSP trees display a high degree of symmetry. In fact, many of the query trees are isomorphic to the majority of SDSP trees, and the majority of SDSP trees have a star shape (i.e. $n-1$ of the vertices in the tree are adjacent to the root). Each non-root vertex in a star tree has the same path name, resulting in a large number of possible candidates per token sequence.

In Table 5.2, we plot the cumulative distribution functions (CDFs) of our experiments. The four Gnutella data sets exhibit a high recovery rate that can be explained by asymmetry and low density. 50% percent of all queries for the p2p-Gnutella08, p2p-Gnutella04, p2p-Gnutella25, p2p-Gnutella30 data sets result in at most 4, 3, 5, 5 candidate query values, respectively. Details of the 50th, 90th, and 99th percentiles can be found in Table 5.1. In Table 5.3 we plot the histograms of the results for QR on the real-world datasets assuming 100% of the queries have been observed.

Random graphs. We also deployed our attack on random graphs, varying the number of nodes ($n = 100, 250, 500, 1000$) and the edge probability ($p = 0.2, 0.4, 0.6, 0.8$). Query recovery was carried out after all possible queries had been issued. For each (n, p) pair we

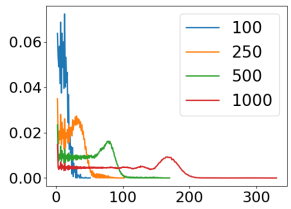
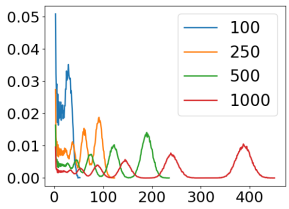
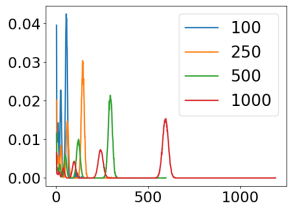
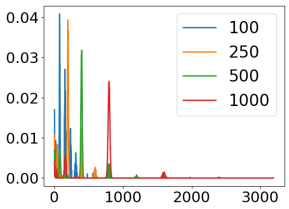
$p = 0.2$	$p = 0.4$	$p = 0.6$	$p = 0.8$
			

Table 5.5: PDFs for QR of random graphs after observing 100% of the queries. On the x axis we plot the number of candidate queries output by our QR attack and on the y axis we plot the percent of total queries.

generated 50 random graphs, encrypted the graphs using the scheme, generated the leakage for all possible SPSP queries, and deployed our query recovery attack on the responses. Then for each recovered multimap, mapping the response to the set of candidate queries, we computed the average number of queries that had each possible number of candidate queries across all 50 graphs. The CDFs of these results can be found in Table 5.4.

In general, we see that an increase in p and n both result in an increase in the size of the maximum query candidate sets. For example, for $n = 100, p = 0.2$ we have that 6.3% of all queries are uniquely recoverable and 50% of all queries are recoverable to at most 10 candidate queries (representing 0.101% of all queries). For $n = 1000, p = 0.2$, we have that 1.5% of all queries are uniquely recoverable and 50% of all queries are recoverable to at most 107 candidate queries (representing 0.0107% of all queries).

As p increases from 0.2 to 0.8, the graphs become more dense and we see a similar trend as seen in the real-world data sets. Denser graphs are closer to complete, and result in more symmetries and larger candidate query sets. As p increases, we also see more "waves" in the CDFs; in the graphs showing the probability density functions (PDFs) (see Table 5.5), these correspond to large clusters of candidate queries all of which have the same path name and hence which cannot be distinguished in a QR attack.

5.5 Conclusion

We have given a query recovery attack against the GKT graph encryption scheme from [52]. The attack model we consider is strong, but it fits within the model used in [52]. The attack begins with an offline preprocessing phase of the graph. In the online phase, our attack waits until it has observed all queries to at least one destination vertex and then outputs a list of candidates for each of these queries. Our attack has the property that the output contains everything that is consistent with the leakage (and nothing more), and always contains the correct query. We have given a precise characterization of when full query recovery is possible. We evaluated our attack against real-world and random graphs.

Another variant of our attack is possible for a network adversary. Such an adversary is assumed to know G but is able to see only the communications between the client and server, that is, search tokens and responses (which, recall, are concatenations of ciphertexts). Note that, for an SPSP query (u, v), the ciphertexts correspond to the vertices that follow u along the path $p_{u,v}$. If the adversary were able to observe the query leakage from all possible queries then it could initialize a graph H and then for each response parse $c_1 \| c_2 \| \dots \| c_t \leftarrow \mathsf{resp}$ and add (c_i, c_{i+1}) to H for $i \in [t-1]$. The components of H then become the trees. These trees are the same as the original query trees, but are missing the leaf nodes. To complete the attack, the adversary would compute the SDSP trees $\{T_v\}_{v \in V}$ and for every $T \in \{T_v\}_{v \in V}$, delete the leaf nodes. Now the adversary can continue with the attack as described in Section 5.3.10, i.e. compute path names and assign the candidate SPSP queries to each resp. This attack would not be able to recover the candidate query values for the leaf nodes of the SDSP trees, so is slightly weaker than our main attack.

Yet a further variant of our attack – and which is an interesting open question – applies when arbitrary subsets of queries have been issued: then the adversary can construct ***partial*** query trees and attempt to identify isomorphic embeddings of them into the SDSP trees.

Our attack leaves open the question of whether other graph encryption schemes can be

similarly attacked. On the constructive side, the question of whether we can build more secure schemes that utilize chaining in a non-interactive manner and which support shortest path queries remains open. Moreover, there is still much work to be done regarding the design of encrypted graph database schemes that can support a variety of queries – an important property for schemes in practical settings. In the next chapter, we present a scheme supporting shortest path queries that provably mitigates this attack and that utilizes classic data-structure techniques to achieve its efficiency.

CHAPTER 6

A GRAPH ENCRYPTION SCHEME FOR SINGLE-PAIR SHORTEST PATH QUERIES WITH LESS LEAKAGE

This is joint work with Esha Ghosh, Kenneth G. Paterson, and Roberto Tamassia.

6.1 Introduction

Graphs are used to model data in numerous large-scale real-world applications ranging from fraud detection to biological networks to recommendation systems. Users often wish to query these graphs for information such as k-nearest neighbors, node clustering, shortest paths/distances between two nodes, and page rank [86]. Plaintext graph databases have consequently been heavily studied in recent years, and many such systems have been developed and widely deployed in industry including Facebook Tao [17], Amazon Neptune [5], GraphDB [99], and Neo4j [97].

The advent of third-party cloud services has resulted in an increase in the outsourcing of data. This data is often sensitive and proprietary, which raises client concerns about privacy. One naive solution would be to encrypt and outsource the whole database, and then download the entire encrypted database each time a query is issued. This is time and bandwidth inefficient and begs the question of how clients can securely outsource their data while enabling query processing on the server-side. An alternative solution would be to use strong cryptographic techniques such as fully-homomorphic encryption [50] or oblivious RAM [54]. While they enable secure computation over encrypted data, these techniques are still expensive at scale despite many recent advances.

To address the issue of efficiently and securely outsourcing graph databases, Chase and Kamara introduced the notion of ***structured encryption (STE)*** [29] which generalizes ***searchable symmetric encryption (SSE)*** [13, 14, 23, 24, 29, 33, 51, 68, 69, 96, 117]. STE

enables the encryption of structured data such that the data can be privately queried in sub-linear time. STE gains its efficiency by sacrificing security to an extent: these schemes leak some information about the queries or underlying data. Security of STE is typically proven using the real-ideal paradigm, which states that the scheme does not leak any information beyond some well-defined leakage function. A long line of work has show that such leakage can be detrimental to the security of schemes supporting key-word search (e.g., [22, 35, 100]) and range queries (e.g.[43, 58, 70, 73–75, 87]), emphasizing the importance of cryptanlyzing proposed schemes.

A ***graph encryption scheme (GES)*** is a form of STE that enables one to encrypt a graph, outsource it to an untrusted server, and then process queries over the encrypted graph. These schemes are efficient, often incurring only a constant overhead over plaintext variants. Our understanding of GESs is in its infancy and existing schemes typically support only a single query type, such as adjacency queries [29], focused subgraph queries [29], approximate shortest distance queries [93], and single-pair shortest path queries [52].

In contrast to standard SSE and range schemes, our understanding of GES security is also limited. Falzon and Paterson [45] demonstrated the first query recovery attack against a GES supporting ***single-pair shortest path (SPSP) queries***. An SPSP query on a graph $G = (V, E)$ takes as input a pair of vertices $(u, v) \in V \times V$ and outputs a shortest path between u and v, if it exists. This attack works against the scheme by Ghosh, Kamara, and Tamassia [52], (henceforth referred to as the ***GKT scheme***) and is optimal in the absence of auxiliary information. It enables an honest-but-curious server with knowledge of the plaintext graph to successfully recover the plaintext queries issued by the client up to some equivalence set which, in practice, can be as small as 3 to 5 queries on real-world social network graphs.

SPSP queries are a fundamental graph query that have numerous applications in social network analysis, routing, resource management, and biology [131]. In this work, we propose a new GES supporting SPSP queries. Our scheme has quantifiably less leakage than the

GKT scheme and mitigates the attack in [45]. The key to our reduced leakage profile is the following. First, we compute the spanning shortest-path trees rooted at each node in the graph that we wish to encrypt. These trees are then decomposed into edge-disjoint paths, which are in turn further processed and stored in an encrypted multimap. Unlike in the GKT scheme, a passive persistent adversary can no longer reconstruct the shortest path trees of G, which was the key to the attack in [45]. Informally, the query reconstruction space of a scheme is the number of unique assignments of plaintext queries to each issued query that produces that same leakage. We prove a number of conditions under which the reconstruction space of our scheme is larger than that of the GKT scheme and show that for many common graph families the gap in the sizes of the reconstruction space is actually exponential in the number of vertices in G.

Our GES achieves asymptotically optimal bandwidth and a further reduced leakage profile at the expense of only a logarithmic factor in storage over the GKT scheme. We support our scheme with a proof of security, a cryptanalysis of the leakage profile, and an implementation to demonstrate the scheme's practicality on real-world datasets. On average, our scheme's response size is similar to that of the GKT scheme, while our query-time is up to 1.65x faster with the potential for additional speed-up through parallelism. In contrast, the GKT scheme's query algorithm cannot be parallelized due to its inherently sequential processing.

6.1.1 Prior work

Schemes. Meng et al. [93] present three schemes of varying efficiency and security that leverage sketch-based distance oracles to support shortest distance queries. Liu et al. [83] and Shen et al. [113] also use distance oracles to pre-compute the shortest distances between queries; the former uses order-preserving encryption for efficiency gains at the expense of leaking the orders of the distances at setup, and the latter only supports constrained shortest path distance queries. Wang et al. [128] utilize additive homomorphic encryption and garbled

Scheme	Scheme Complexity			Attack		Size of Query Reconstruction Space		
	Resp size	Query time	Space	Citation	# Queries	K_n	L_n	$S_{[n]}$
GKT [52]	t	t	n^2	[45]	n^2	$(n!)^{n+1}$	$2^{n/2}$	1
This work	$2t$	$\log n + 2t$	$n^2 \log n$			$(n^2)!$	$\geq (n!)^{\log n - 1}$	$\geq \prod_{i=1}^{\ell} \left(\prod_{j=1, j\neq i}^{\ell} \frac{k_j}{2}! \right)^{k_j}$

Table 6.1: Comparison of our scheme and the GKT scheme [52], both of which are GES that support SPSP queries. Here, n denotes the number of vertices of the graph and t the length of the shortest path returned by a query. K_n denotes the complete graph on n vertices, L_n the line on n vertices, and S_n the "asymmetric" star with one central node and n incident paths of lengths $1, 2 \ldots, n$, respectively. Proofs of the query reconstruction space bounds can be found in Section 6.5.

circuits to encrypt graph data and support shortest distance queries; their scheme additionally relies on a third party, the proxy. Ghosh et al. [52] describe a scheme based on the SP-matrix of the graph and whose setup time, query time, and storage are asymptotically optimal.

A number of STE schemes supporting other query types and graphs have been described such as GES supporting top-k-nearest neighbors [82], STE for conceptual graphs [105], STE for knowledge graphs [130], and dynamic STE for bipartite graphs [76, 80]. Solutions for graph queries in other security models have also been proposed, but are outside the scope of this work. These approaches include SGX [39], private information retrieval [129], differential privacy [111], and structural anonymization [21].

Attacks. The first leakage abuse attack against a GES was described by Goetschmann [53]. This attack succeeds against one of the schemes by Meng et al. [93] that supports approximate shortest path distance queries; Goetschmann describes two methods for estimating distances between the nodes of the graph and then uses these methods to infer the plaintext vertices in each issued query. Most recently, Falzon and Paterson [45] give a query recovery attack against the GKT scheme [52]. They show that a server-side adversary with knowledge of the plaintext graph G and the leakage of certain queries can recover the set of possible plaintext queries for each issued query.

6.1.2 Contributions

Throughout this work, we let $G = (V, E)$ denote a graph, $n = |V|$ the number of vertices, and $m = |E|$ the number of edges. A single pair shortest-path query on a graph can be answered on the fly by using a single-source shortest path (SSSP) algorithm, such as Dijkstra's algorithm which runs in time $O(m + n \log n)$. This approach, however, is very costly when multiple queries are processed on the graph. To avoid such overhead, one must pre-process the graph and store the shortest path information in a data structure that can then be used to answer SPSP queries efficiently. Examples of such data structures include the SP-matrix [32] and distance oracles [31].

Existing attacks work against GESs that use either the SP-matrix or distance oracles. We thus propose a new data structure based on classical graph algorithm techniques to build a scheme that provably mitigates these attacks. Let $G = (V, E)$ be the graph that we wish to encrypt. At a high level, our GES works as follows. For each vertex $v \in V$ we compute the shortest path tree rooted at v whose paths correspond to shortest paths in G; we denote this tree as T_v. We decompose each tree T_v into edge-disjoint paths. Though one could use any algorithm for decomposing the trees, we opt for the ***heavy-light decomposition*** for its guarantees on bounding the number of sub-paths any path is decomposed into.

We could stop here and store the disjoint paths. However, in the worst case, the longest path returned might still have length $O(n)$. Given a disjoint path p of length ℓ, we extract subpaths of p of lengths that are powers of two i.e. $2^0, 2^1, \ldots, 2^{\lceil \log \ell \rceil}$. We call these sub-paths canonical fragments. These fragments are then stored in a multimap. A second multimap stores the map between SPSP queries and the identifiers of the fragments needed to reconstruct the shortest path for the given query. The two multimaps are then encrypted using a standard encrypted multimap scheme.

To query the graph, the server computes one lookup to learn the identifiers of the fragments and a second lookup to retrieve the actual fragments. This is done non-interactively, i.e. in

one round trip between the client and server, since we use a response-revealing EMM to encrypt the first multimap. Our approach is bandwidth and storage efficient, and mitigates the attack from prior work.

Our contributions can be summarized as follows:

- We present a new non-interactive graph encryption scheme that supports single-pair shortest path queries. Our scheme is based on a novel data structure that stores sub-paths of lengths powers of two to minimize bandwidth and reducing leakage. (Section 6.4)

- We describe a new data structure for responding to SPSP queries. This data structure contributes to the enhanced security of our scheme whilst maintaining optimal bandwidth complexity and incurring only an additional logarithmic factor in storage overhead. (Section 6.4)

- We support our scheme with a thorough cryptanalysis and prove that there is an information-theoretic gap between the size of the query reconstruction space of our scheme and that of the GKT scheme for several graph families.(Section 6.5)

- We implement our scheme and test its performance on a number of real-world datasets, including ones of social networks and geographical networks. We show improved round-trip query time over the GKT scheme. (Section 6.6)

6.2 Preliminaries

Notation. For a integer n, let $[n] = \{1, 2, \ldots, n\}$. We denote concatenation of two strings a and b as $a||b$. We denote use $\varnothing$ or $\perp$ to refer to the empty set. Given a set X we use $x \leftarrow\!\$\, X$ to denote that the element x was sampled uniformly at random from X. We denote a function negligible in λ by $\mathsf{negl}(\lambda)$. Unless otherwise stated, we simply write $\log n$ for $\log_2 n$.

Graphs. A ***graph*** $G = (V, E)$ is a tuple comprised of a set of vertices V (of size n) and a set of edges $E = V \times V$ (of size m). For simplicity of presentation, we assume that all graphs we consider are connected. However our scheme can be used to encrypt multi-component graphs without any modifications.

In this work, we consider GES that support ***single pair shortest path (SPSP) queries*** on a graph $G = (V, E)$. An SPSP query is the evaluation of a function SPSP that takes as input two vertices $u, v \in V$, and outputs a path $p_{u,v} = (u, w_1, \ldots, w_t, v)$ of minimal length in G if u and v are connected, and outputs $\perp$ otherwise.

6.2.1 Graph Encryption Scheme

Definition 20. *([29]) A* ***graph encryption scheme (GES)*** *is a tuple of algorithms* $\mathsf{GES} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Token}, \mathsf{Search}, \mathsf{Reveal})$ *with the following syntax:*

- KeyGen *is a probabilistic algorithm that takes as input a security parameter* λ *and outputs a secret key* K*.*
- $\mathsf{Encrypt}$ *is a probabilistic algorithm that takes a secret key* K *and a graph* G *and outputs an encrypted database* EDB*.*
- Token *takes as input a key* K *and query* q *and returns a search token* tk*.*
- Search *takes as input an encrypted database* EDB *and a search token* tk *and outputs a response* resp*.*
- Reveal *takes as input a key* K *and response* resp *and outputs plaintext* m*.*

Note that this is a purely syntactical definition and has no mention of client or server. In practice, $\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Token}$, and Reveal would be executed by the client while Search would be executed by the server.

Correctness. A graph encryption scheme for SPSP queries GES is correct if for all graphs $G = (V, E)$ and all queries $q = (u, v) \in V \times V$, if $K \leftarrow \mathsf{GES.KeyGen}(1^\lambda)$ and $\mathsf{EDB} \leftarrow \mathsf{GES.Encrypt}(K, G)$, then the resulting $m \leftarrow \mathsf{GES.Reveal}(K, \mathsf{resp})$ is a shortest path from u to v in G where $\mathsf{tk} \leftarrow \mathsf{GES.Token}(K, q)$ and $\mathsf{GES.Search}(\mathsf{EDB}, \mathsf{tk})$.

Security. The security of GESs is parameterized by a leakage function $\mathcal{L} = (\mathcal{L}_\mathsf{S}, \mathcal{L}_\mathsf{Q})$ that specifies an upper bound on the information leaked at setup ($\mathcal{L}_\mathsf{S}$) and at query time ($\mathcal{L}_\mathsf{Q}$). We define security using the real-ideal paradigm with respect to passive persistent adversaries who execute the protocol honestly and who learn the output of the leakage functions. Such adversaries may include an honest-but-curious server or a network adversary that has compromised the communication channel.

Definition 21. *Let* $\mathsf{GES} = (\mathsf{Encrypt}, \mathsf{Query})$ *be a graph encryption scheme and let* $\mathcal{L} = (\mathcal{L}_\mathsf{S}, \mathcal{L}_\mathsf{Q})$ *be a tuple of stateful algorithms. We say that* GES *is* $\mathcal{L}$***-secure*** *if for all polynomial-time adversaries* $\mathcal{A}$*, there exists a polynomial-time simulator* $\mathcal{S}$ *such that*

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

and games $\mathbf{Real}_\mathcal{A}^\mathsf{GES}(\lambda)$ *and* $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^\mathsf{GES}(\lambda)$ *are defined as in Figure 6.1.*

6.2.2 Encrypted Multimap Scheme

Encrypted multimap schemes are fundamental building blocks of our GES. EMM schemes can be response-revealing or response-hiding. Informally, a response-revealing scheme reveals the plaintext answer of a query to the server whereas a response-hiding scheme does not.

Definition 22. *A* ***response-hiding encrypted multimap (EMM) scheme*** *is a tuple of algorithms* $\mathsf{EMM\text{-}RH} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Token}, \mathsf{Get}, \mathsf{Reveal})$ *with the following syntax:*

- KeyGen *is probabilistic and takes a security parameter* λ*, and outputs a secret key* K*.*

$\mathbf{Real}^{\mathsf{GES}}_{\mathcal{A}}(1^\lambda)$	$\mathbf{Ideal}^{\mathsf{GES}}_{\mathcal{A},\mathcal{S}}(1^\lambda)$
INITALIZE(D)	INITALIZE(D)
1 : $K \leftarrow \mathsf{GES.KeyGen}(\lambda)$	1 : $(\alpha, \mathsf{st}_{\mathcal{L}}) \leftarrow \mathcal{L}_{\mathsf{S}}(\mathsf{D})$
2 : $\mathsf{EDB} \leftarrow \mathsf{GES.Encrypt}(K, G)$	2 : $(\mathsf{ED}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}(\alpha)$
3 : **return** EDB	3 : **return** EDB
QUERY(q)	QUERY(q)
4 : $\mathsf{tk} \leftarrow \mathsf{GES.Token}(K, q)$	4 : $(\alpha, \mathsf{st}_{\mathcal{L}}) \leftarrow \mathcal{L}_{\mathsf{Q}}(q, \mathsf{st}_{\mathcal{L}})$
5 : $\mathsf{resp} \leftarrow \mathsf{GES.Search}(\mathsf{EDB}, \mathsf{tk})$	5 : $(\mathsf{resp}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}(\alpha, \mathsf{st}_{\mathcal{S}})$
6 : **return** resp	6 : **return** resp
FINALIZE(b)	FINALIZE(b)
7 : **return** b	7 : **return** b

Figure 6.1: Games $\mathbf{Real}^{\mathsf{GES}}_{\mathcal{A}}$ and $\mathbf{Ideal}^{\mathsf{GES}}_{\mathcal{A},\mathcal{S}}$.

- Encrypt *is probabilistic and takes as input a secret key K and multimap* M*, and outputs an encrypted multimap* $\mathbf{EM}$.
- Token *takes a key K and a label* lab*, and outputs a search token* tk.
- Get *takes a search token* tk *and an encrypted multimap* $\mathbf{EM}$ *and returns response* resp.
- Reveal *takes key K and response* resp *and returns a set of plaintext values* $\{\mathsf{val}_i\}_{i\in[k]}$.

A ***response-revealing EMM scheme*** comprises of a tuple of only four algorithms $\mathsf{EMM\text{-}RR} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Token}, \mathsf{Get})$ such that $\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Token}$ are as in Definition 22 and Get instead takes as input enccrypted database EDB and search token tk and returns a set of plaintext values $\{\mathsf{val}_i\}_{i\in[k]}$. Our graph encryption scheme is response-hiding and makes black-box use of one response-revealing and one response-hiding EMM scheme.

Correctness. Correctness of EMM requires that for all multimaps M and labels lab in M, if $K \leftarrow \mathsf{EMM.KeyGen}(\lambda)$, $\mathbf{EM} \leftarrow \mathsf{EMM.Encrypt}(K, \mathsf{M})$, then executing $\mathsf{EMM.Reveal}(K, \mathsf{resp})$ such that $\mathsf{tk} \leftarrow \mathsf{EMM.Token}(K, \mathsf{lab})$ and $\mathsf{resp} \leftarrow \mathsf{EMM.Get}(\mathbf{EM}, \mathsf{tk})$, returns $\mathsf{M}[\mathsf{lab}]$.

Security. Security of an EMM scheme is also defined using the real-ideal paradigm. The $\mathbf{Real}^{\mathsf{EMM}}_{\mathcal{A}}$ and $\mathbf{Ideal}^{\mathsf{EMM}}_{\mathcal{A}}$ games are the same as in Figure 6.1, except that INITIALIZE is

executed using $\mathsf{EMM.KeyGen}$ and $\mathsf{EMM.Encrypt}$, and $\textsc{Query}$ is executed using $\mathsf{EMM.Token}$ and $\mathsf{EMM.Get}$.

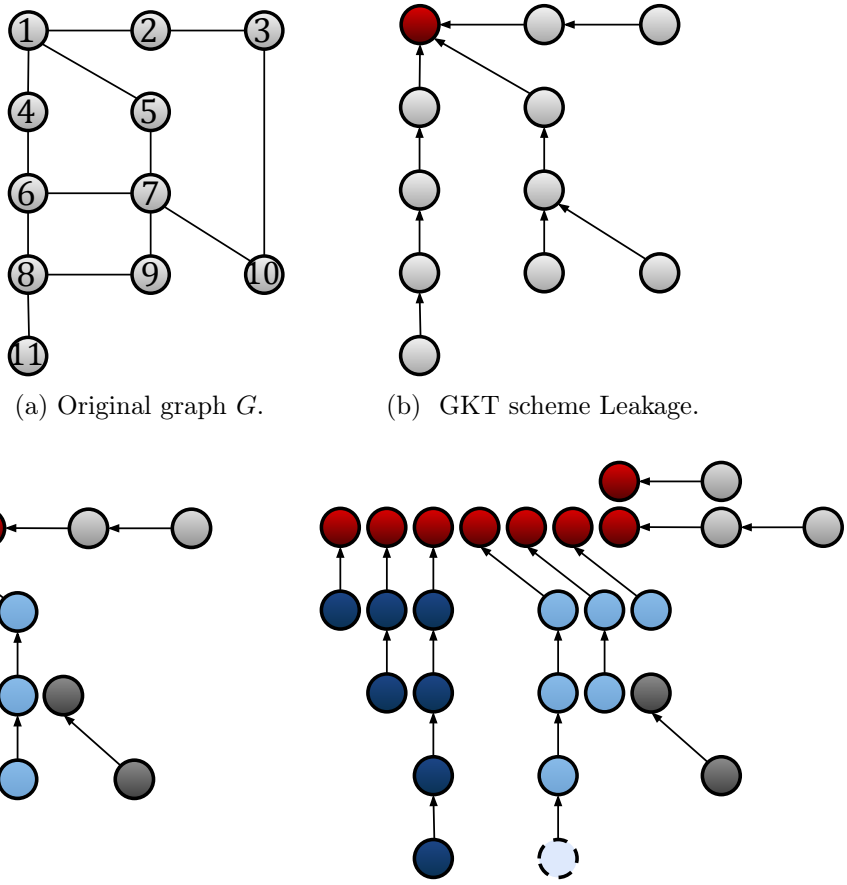
Leakage Functions. In our work we consider EMM schemes with the following standard leakage functions.

- ***Multimap size*** (Size): This setup leakage refers to the total number of values in the multi-map. Formally, for a multimap M with label space $\mathbb{L}$, $\mathsf{Size}(\mathsf{M}) = \sum_{\mathsf{lab} \in \mathbb{L}} |\mathsf{M}[\mathsf{lab}]|$.

- ***Query pattern*** (QP): Reveals whether two queries are equal. Let M be a multimap and $\mathsf{lab}_1, \ldots, \mathsf{lab}_k$ be a sequence of queries, then $\mathsf{QP}(\mathsf{M}, (\mathsf{lab}_1, \ldots, \mathsf{lab}_k)) = M$ where M is a $k \times k$ matrix of bits such that $M[i, j] = 1$ if and only if $\mathsf{lab}_i = \mathsf{lab}_j$.

- ***Access pattern*** (AP): Reveals the individual values returned for each query. Let M be a multimap and $\mathsf{lab}_1, \ldots, \mathsf{lab}_k$ be a sequence of queries, then $\mathsf{AP}(\mathsf{M}, (\mathsf{lab}_1, \ldots, \mathsf{lab}_k)) = (\mathsf{M}[\mathsf{lab}_i])_{i \in [k]}$.

- ***Volume pattern*** (Vol): Reveals the number of values returned. Let M be a multimap and $\mathsf{lab}_1, \ldots, \mathsf{lab}_k$ be a sequence of queries, then $\mathsf{Vol}(\mathsf{M}, (\mathsf{lab}_1, \ldots, \mathsf{lab}_k)) = (|\mathsf{M}[\mathsf{lab}_i]|)_{i \in [k]}$.



(a) Original graph G.

(b) GKT scheme Leakage.

(c) Scheme 1 Leakage.

(d) Scheme 2 Leakage.

Figure 6.2: (a) The original graph G and (b) its corresponding SDSP tree rooted at 1. The GKT scheme leaks the entire topology of the SDSP trees. In contrast, our schemes only leak edge-disjoint paths. In particular our first scheme leaks (c) the set of disjoint paths comprising the queried path, respectively. Our second scheme leaks (d) the set of minimal-length canonical fragments comprising the queried path, these fragments may contain padding to pad lengths up to the next largest power of two. Padding vertices are depicted with a dotted border. Note that the same set of paths or canonical fragments may correspond to distinct queries.

6.3 Technical Background

We now introduce the technical background for our scheme. The leakage profile of the original GKT scheme leaked the edges along which two paths with the same destination vertex intersect at. This leakage can be detrimental to the security of the queries and, in fact, Falzon and Paterson [45] demonstrate how this leakage can be leveraged to recover the plaintext queries. Our goal is to thus reduce the leakage.

Let $G = (V, E)$ be the graph that we wish to encrypt. To achieve this reduced leakage profile, we first compute the ***all-pairs shortest paths (APSP)*** of G. We show that for common APSP algorithms, such as Floyd-Warshall [47], taking the union of paths with the same destination vertex results in n rooted spanning trees $\{T_v\}_{v \in V}$: each tree T_v is rooted at a different vertex $v \in V$ and the paths correspond to shortest paths in G. These trees are then decomposed into edge-disjoint paths. Our scheme can be instantiated using ***any*** algorithm that decomposes trees into edge-disjoint paths. However, to concretize our schemes we employ a data-structure technique called ***heavy-light decomposition (HLD)*** [116] to decompose the trees. In particular, we use HLD for its efficiency guarantees and describe the algorithm in detail in Section 2.

We start by introducing some notation, and proving that many common all-pairs shortest path (APSP) algorithms output collections of paths from which spanning rooted shortest path trees can be computed. This is a necessary condition, since we must apply HLD to a rooted tree.

Definition 23. *Let $G = (V, E)$ be a graph and P be a collection of paths of G. The **graph induced by** P is defined as $G|_P = (V_P, E_P)$ where $V_P = \{v : v \in e, e \in p, p \in P\}$ and $E_P = \{e : e \in p, e \subseteq P\}$. By definition, $V_P \subseteq V$ and $E_P \subseteq E$.*

Lemma 6.3.1. *Let $G = (V, E)$ be a graph and P be the collection of paths that results from*

running Floyd-Warshall on G. [1] *Let P_v denote the set of paths in P that terminate at v. Then for all $v \in V$, the graph $G|_{P_v}$ forms a tree with v as its root.*

Proof. We must prove two properties: (1) $G|_{P_v}$ is connected and (2) there are no cycles in $G|_{P_v}$. By definition of P_v, all paths must end at v. Thus any two vertices $u, u' \in V_{P_v}$ are connected through an undirected path via v and property (1) holds.

We now prove property (2). Let the vertices in V be labeled from 1 to n and let $A(u, v, k)$ denote the shortest path from u to v using only vertices 1 to k. Recall the recurrence used by Floyd-Warshall.

$$A(u, v, k) = \min \begin{cases} A(u, v, k-1) \\ \\ A(u, k, k-1) + A(k, v, k-1) \end{cases}$$

where $A(u, v, 0) = (u, v)$. For the sake of contradiction, suppose that the recurrence produces a cycle in $G|_{P_v}$. Then there must be two distinct paths between a pair of vertices in P_v as depicted in Figure 6.3.

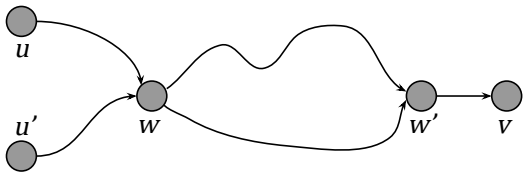


Figure 6.3: A cycle in P_v.

As k iterates over $[n]$, we update the shortest path from u to v using only vertices $1, \ldots, k$. Similarly for the shortest path from u' to v. However, a cycle as in Figure 6.3 would imply that for $k = w$ we have $A(u, v, k) = A(u, k, k-1) + A(k, v, k-1)$ and $A(u', v, k) = A(u', k, k-1) + A(k, v, k-1)$. Yet the path corresponding to $A(k, v, k-1)$ in the two equations must have been distinct. This is a contradiction. Thus P_v cannot contain a cycle. □

1. We prove this for the Floyd-Warshall algorithm, but this property holds for other similar APSP algorithms.

Corollary 6.3.2. *Let $G = (V, E)$ be a graph and P the collection of paths resulting from running Floyd-Warshall on G. For any vertex $v \in V$, if two paths $p, p' \in P_v$ coincide at a vertex u, then they must coincide at each vertex along the path from u to v.*

For convenience, we denote tree $G|_{P_v}$ as T_v and refer to it as the ***single destination shortest path (SDSP) tree*** for v in G.

After computing all SDSP trees, our scheme decomposes each tree into edge-disjoint paths using HLD. At a high level, HLD works by labeling edges in a tree T as either "heavy" or "light" such that the light edges demarcate where an edge-disjoint path ends and a new path starts. HLD additionally guarantees that any path in T crosses no more than $\log n$ disjoint paths. Applying HLD to a tree may, in the worst case, result in an edge-disjoint path of length $O(n)$. This happens when most edges are heavy e.g. when G is a path. To address this, our scheme computes and stores what we call the ***canonical fragments*** of each edge-disjoint path. The canonical fragments of a path $p_{u,v}$ are the subpaths ending at v whose lengths are powers of two. This allows us to optimize bandwidth at the expense of a small additional overhead in storage.

Definition 24. *Let $p_{u,v}$ be a path padded up to the next smallest power of 2 such that the padding vertices are pre-pended to u. Let $p_{u,v}^{(j)}$ denote the subpath comprised of the last 2^j edges in $p_{u,v}$. The **canonical fragments** of $p_{u,v}$ are $\{p_{u,v}^{(j)} : 0 \leq j \leq \log_2 |p_{u,v}|\}$.*

6.4 A GES With Less Leakage

In this section, we present OurGES, a scheme that leverages indirection to ensure non-interactivity and reduce leakage. Our scheme has optimal bandwidth complexity and has a reduced leakage profile as compared to GKT [52].

$\mathsf{KeyGen}(1^\lambda) \to K$
  $K_1 \leftarrow \mathsf{EMM\text{-}RR.KeyGen}(1^\lambda)$
  $K_2 \leftarrow \mathsf{EMM\text{-}RH.KeyGen}(1^\lambda)$
  return $K_1 \| K_2$

$\mathsf{Encrypt}(K, G) \to \mathsf{EDB}$
  Initialize multimaps M_1 and M_2
  Parse $K_1 \| K_2 \leftarrow K$
  for $r \in V$ **do**
    Compute SDSP tree T_r rooted at r in G
    $T_r^A \leftarrow \textsc{ComputeHld}(T_r, r)$
    for each subpath $p_{u,v} \in T_r^A$ in BFS manner **do**
      Let ℓ be the next power of 2 greater than $|p_{u,v}|$
      Pad $p_{u,v}$ to length ℓ
      for $j \in [0, \lceil \log_2 \ell \rceil]$ **do**
        // Compute fragment and add it to M_2.
        Let $p_{u,v}^{(j)}$ be the path of the last 2^j edges of $p_{u,v}$
        $\mathsf{M}_2[(r, u, v, j)] \leftarrow p_{u,v}^{(j)}$
        if $j = 0$ **then** $s = p_{u,v}^{(0)}$ **else** $s = p_{u,v}^{(j)} \setminus p_{u,v}^{(j-1)}$
        for non-pad vertex w in s **do**
          // Add tokenset of query (w, r) to M_1
          $\mathsf{tk} \leftarrow \mathsf{EMM\text{-}RH.Token}(K_2, (r, u, v, j))$
          $\mathsf{M}_1[(w, r)] \leftarrow \mathsf{M}_1[(v, r)] \cup \{\mathsf{tk}\}$
          Permute $\mathsf{M}_1[(w, r)]$
  Pad M_1 and M_2 up to $n^2 \log n$ and $4n^2$, respectively.
  $\mathbf{EM}_1 \leftarrow \mathsf{EMM\text{-}RR.Encrypt}(K_1, \mathsf{M}_1)$
  $\mathbf{EM}_2 \leftarrow \mathsf{EMM\text{-}RH.Encrypt}(K_2, \mathsf{M}_2)$
  return $(\mathbf{EM}_1, \mathbf{EM}_2)$

Figure 6.4: Psuedocode for our scheme, $\mathsf{OurGES.Encrypt}$, which supports single pair shortest path queries over an encrypted graph.

6.4.1 Scheme Description

High-level Description. Given a graph $G = (V, E)$, the client computes the set of SDSP trees $\{T_v\}_{v \in V}$ and decomposes each tree into disjoint paths (e.g. using HLD). The idea is to encode the shortest path information as a set of fragments which can be retrieved using a set of look-ups to two multimaps. Multimap M_1 maps each query to a set of search tokens used to look up the corresponding canonical fragments in the second multimap; it is encrypted using a response-revealing EMM scheme. Multimap M_2 maps each canonical fragment identifier to the respective canonical fragment; it is encrypted using a response-hiding EMM scheme. We denote the encrypted versions of M_1 and M_2 as $\mathbf{EM}_1$ and $\mathbf{EM}_2$, respectively.

```
// Compute search token.
Token_C(K, (u, v)) → tk
    Parse K_1‖K_2 ← K
    tk ← EMM-RR.Token(K_1, (u, v))
    return tk
// Look up encrypted fragments.
Search_S(EDB, tk) → resp
    Initialize resp ←⊥
    Parse (EM_1, EM_2) ← EDB
    T ← EMM-RR.Get(EM_1, tk)
    for tk' ∈ T do
        c ← EMM-RH.Get(EM_2, tk')
        resp ← resp ∪ {c}
    return resp
// Decrypt and recover path.
Reveal_C(K, resp) → p
    Parse K_1‖K_2 ← K
    Initialize P ← ∅
    for c ∈ resp do
        m ← EMM-RH.Reveal(K_2, c)
        Unpad m
        P ← P ∪ {m}
    Sort P into path p from u to v
    return p
```

Figure 6.5: Psuedocode for our scheme's query protocol, which comprises of three algorithms: OurGES.Token, OurGES.Search, and OurGES.Reveal.

To issue query $(u, v) \in V \times V$, the client computes a query-specific search token $\mathsf{tk}_{u,v}$ which the server uses to retrieve the token set T from $\mathbf{EM}_1$. For each token $\mathsf{tk}' \in T$, the server retrieves the corresponding canonical fragment from $\mathbf{EM}_2$ and adds it to the response. By property of HLD, each shortest path is comprised of no more than $\log n$ fragments and, thus, search requires at most $\log n$ look-ups to $\mathbf{EM}_2$ (one look-up for each search token obtained from $\mathbf{EM}_1$) to retreieve the at most $2t$ edges. The server returns the set of encrypted fragments to the client who then decrypts them to recover the shortest path.

Formal description. KeyGen takes as input a security parameter λ and returns a pair of keys $K_1 \| K_2$, one each for the response-revealing and the response-hiding EMM schemes.

Encrypt takes as input key $K = K_1 \| K_2$ and the graph $G = (V, E)$ that the client wishes to encrypt. It initializes empty multimaps M_1 and M_2. For each vertex $r \in V$ it computes

the SDSP tree rooted at r to obtain T_r. For each T_r, the client decomposes the tree into edge-disjoint paths e.g. using the HLD algorithm (Algorithm 2). We denote the decomposed tree as T_r^A.

Importantly, the disjoint paths are processed in a breadth first search (BFS) manner, so that the multimap can be computed in one traversal of each tree. For each path $p_{u,v}$ in T_r^A, as it is discovered, and for each canonical fragment $p_{u,v}^{(j)}$ of $p_{u,v}$, the client:

1. Generates token $\mathsf{tk}' \leftarrow \mathsf{EMM\text{-}RH.Token}(K_2, (r, u, v, j))$ using the response-hiding EMM;
2. Sets $\mathsf{M}_2[(r, u, v, j)] \leftarrow p_{u,v}^{(j)}$;
3. Sets $\mathsf{M}_1[(w, r)] \leftarrow \mathsf{M}_1[(u, r)] \cup \{\mathsf{tk}'\}$.

The key is that the fragments needed to reconstruct the shortest path from u to r comprises the fragment from u to v, and the fragments of the shortest path from v to r. Since T_r^A is processed in a BFS manner, $\mathsf{M}[(r, u)]$ has already been computed.

The client pads M_1 up to $n^2 \log n$ and M_2 up to $4n^2$. Then it encrypts the multimaps i.e., $\mathbf{EM}_1 \leftarrow \mathsf{EMM\text{-}RR.Encrypt}(K_1, \mathsf{M}_1)$ and $\mathbf{EM}_2 \leftarrow \mathsf{EMM\text{-}RH.Encrypt}(K_2, \mathsf{M}_1)$, and the encrypted graph $(\mathbf{EM}_1, \mathbf{EM}_2)$ is outsourced to the server. The padding step is necessary to prevent leaking information about the graph at setup (See Section 6.4.2 for a discussion).

To query for $(u, v) \in V \times V$, the client computes a token $\mathsf{tk}_{u,v} \leftarrow \mathsf{EMM\text{-}RR.Token}(K_1, (u, v))$ which it sends to the server. The server uses $\mathsf{tk}_{u,v}$ to look up the token set T in $\mathbf{EM}_1$ that is needed to retrieve the corresponding canonical fragments in $\mathbf{EM}_2$. The server initializes empty set resp and for each $\mathsf{tk}' \in T$ adds the encrypted fragment $c \leftarrow \mathsf{EMM\text{-}RH.Get}(\mathbf{EM}_2, \mathsf{tk}')$ to resp. Finally, resp is returned to the client who can then decrypt and sort the fragments to recover the shortest path.

The pseudocode for OurGES can be found in Figures 6.4 and 6.5.

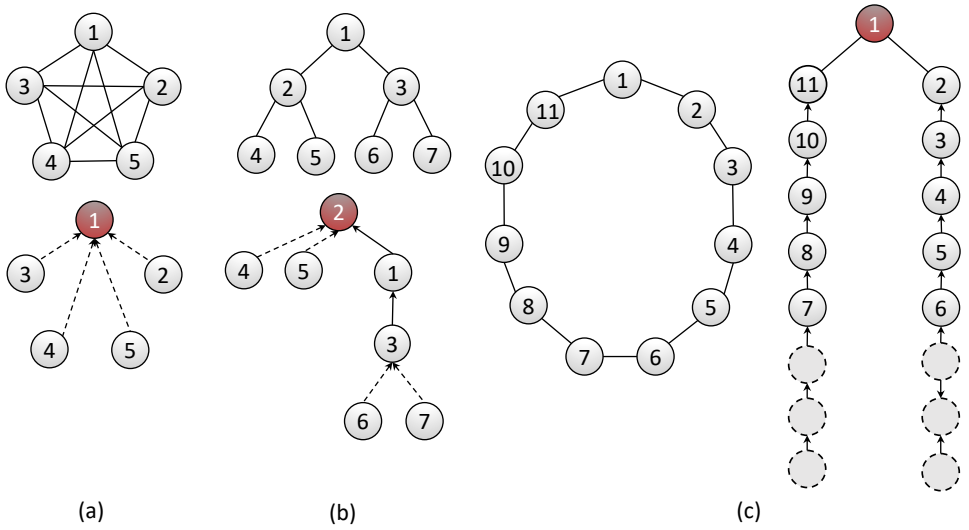


Figure 6.6: Examples of graphs that, without padding the resulting multimaps, would achieve asymptotically different sizes for M_1 and M_2. Solid and dashed edges denote heavy and light edges, respectively. Dotted circles denote padding vertices.

6.4.2 Complexity and Correctness

Encrypt is upper bounded by the time needed to compute the SDSP trees, which for general graphs takes time $O(n^3)$. Token entails a single call to the EMM-RR.Token. Our chosen implementation of EMM-RH only requires two PRF evaluations [23] which take time $O(1)$. Assuming use of HLD to decompose the trees into disjoint subpaths, Search takes time $O(\log n + t)$, since any shortest path results in at most $1 + \log n$ look-ups to the EMMs to retrieve at most $2t$ fragments. In the worst case the server must return $2t$ edges, where t is the length of the shortest path and so client-side decryption, Reveal, takes time $O(2t)$.

The first multimap takes $n^2 \log n$ space and the second multimap takes $4n^2$ space, for a total storage complexity of $O(n^2 \log n)$. To motivate the need for padding the multimaps, we describe some pathological graphs for which the sizes of M_1 and M_2 vary wildly without the padding step in line 24 (Figure 6.4).

Upperbound on M_1. Let G be the complete graph on n vertices. Each SDSP tree of G is a star: a tree whose root r is adjacent to each of the other $n-1$ vertices. Computing the HLD of a star results in n edge-disjoint paths all of length $1 = 2^0$ (See Figure 6.6a). Each query

corresponds to exactly one fragment of size 1, and without padding the multimaps, the size of M_1 would be n^2. In contrast, if G is a balanced binary tree then any SDSP tree T_r of G would itself be a binary tree. In particular, each T_r contains a balanced binary sub-tree on $n/2$ vertices. Recall that applying HLD to a balanced binary tree results in all light edges (see Figure 6.6b). Thus in each SDSP tree, there are at least $n/4$ leaf nodes whose shortest paths to r comprise $\log(n/4)$ disjoint paths and the resulting M_1, without padding the multimaps, would be of size $O(n^2 \log n)$.

Lemma 6.4.1. *Let $\lambda \in \mathbb{Z}$ and G a graph on n vertices. Let $K \leftarrow \mathsf{OurGES.KeyGen}(1^\lambda)$ and $(\mathbf{EM}_1, \mathbf{EM}_2) \leftarrow \mathsf{OurGES.Encrypt}(K, G)$. Then at the end of line 11 in the* Encrypt *algorithm, $|\mathsf{M}_1| \leq n^2 \log n$.*

Proof. M_1 maps each SDSP query to a token set, such that each token in the set corresponds to a fragment. By Theorem 2.1.1, the shortest path from any vertex to the root of an annotated tree T_r^A crosses at most $\lfloor \log n \rfloor$ light edges. A shortest path from any vertex u to the root r in T_r^A comprises of no more than $\log n$ disjoint paths and, thus, each query corresponds to at most $\log n$ fragments. There are $n^2 - n$ possible SPSP queries. Thus, at the end of line 11 (Figure 6.4), M_1 has size at most $n^2 \log n$. □

In Figure 6.7, we plot the ratio of the size of M_1 (prior to padding) when encrypting a balanced binary tree to our upperbound of $O(n^2 \log n)$. We see that out ratio closely approaches 0.8.

Upperbound on M_2. To see why we need to pad M_2 up to $4n^2$ observe the following. Consider again the complete graph on n vertices and its resulting SDSP trees. Since the fragments of these n trees are trivial, then the size of M_2, prior to padding, would be n^2. On the other hand, if G is a cycle of size $n = 2(2^k + 1) + 1$, then every SDSP tree T_r will comprise a root r from which two paths of length $2^k + 1$ descend (see Figure 6.6c). The subtree induced by r, one of r's children, and its descendants forms a path p of length $2^k + 1$

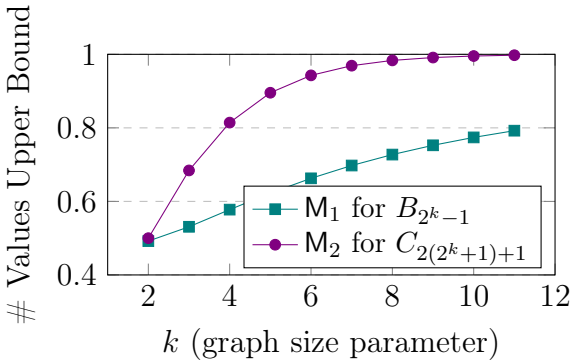


Figure 6.7: We depict the number of values in M_1 prior to padding when encrypting the balanced binary tree on $n = 2^k - 1$ nodes to the bound $n^2 \log n$ (—■—). We also depict the number of values in M_2 prior to padding when encrypting the cycle graph on $n = 2(2^k + 1) + 1$ nodes to the bound $4n^2$ (—●—). Observe that our worst-case examples approach our theoretical limits to within a small constant factor.

and on line 13 the client will pad the path up to 2^{k+1}, effectively doubling the length of the path. Storing each of the canonical fragments of p adds an additional factor of 2 overhead. This yields a multimap M_2 whose size approaches $4n^2$ as n tends to infinity.

Lemma 6.4.2. *Let $\lambda \in \mathbb{Z}$ and $G = (V, E)$ be any graph on n vertices. Let $K \leftarrow \mathsf{OurGES.KeyGen}(1^\lambda)$ and $(\mathbf{EM}_1, \mathbf{EM}_2) \leftarrow \mathsf{OurGES.Encrypt}(K, G)$. Then at the end of line 11 in the* Encrypt *algorithm, $|\mathsf{M}_2| \leq 4n^2$.*

Proof. On line 13 (Figure 6.4), each edge-disjoint path of length ℓ is padded to the next power of two. In the worst case $\ell = 2^k + 1$ for an integer k, which results in padding the path up to $2 \cdot 2^k$. The length of the padded path approaches 2ℓ as k tends to infinity. Storing the canonical fragments of each path requires an additional 2^{k+1} storage. M_2 maps fragment identifiers to fragments. Thus, at the end of line 11, M_2 has size at most $4n^2$. □

In Figure 6.7, we also plot the ratio of the size of M_2 (prior to padding) when encrypting a cycle graph to our upperbound of $O(4n^2)$. We see that the ratio approaches 1, thus proving that our asymptotic bound is tight.

Summary. We now present a theorem that summarizes the correctness and complexity.

Theorem 6.4.3. *Let $G = (V, E)$ be a graph such that $n = |V|$. Let* EDB *be the result of encrypting G using* OurGES*. Executing* Encrypt *on G takes $O(n^3)$ time and produces an encrypted database of size $O(n^2 \log n)$. Executing* Query *on* EDB *and SPSP query $(u, r) \in V \times V$ results in the shortest path from u to r having a response size of at most $2t$, where t is the length of the shortest path from u to r.* Token *runs in time $O(1)$ and produces a token of size $O(1)$.* Search *runs in $O(\log n + t)$ time and* Reveal *runs in $O(t)$ time.*

Proof. First we show that the response is the shortest path from u to r. Let T_r^A be the annotated SDSP tree rooted at r. We proceed by induction on the disjoint subpaths as they are discovered using BFS. Let $p_{v_0,r}$ be the first disjoint path, and let $\{p_{v_0,r}^{(j)}\}_{j\in[0,\ell]}$ be its set of canonical fragments. Then for every $j \in [1, \ell]$ and every non-pad vertex w in $p_{v_0,r}^{(j)} \setminus p_{v_0,r}^{(j-1)}$ it is easy to see that $\mathsf{M}_1[(w, r)]$ corresponds to the fragment needed to recover the path from w to r. Suppose that this also holds true for the first $k - 1$ subpaths. Let $p_{u,v}$ be the k-th path discovered. By the inductive hypothesis and correctness of BFS, then $\mathsf{M}_1[(u, r)]$ must have been computed correctly. Thus for every $j \in [1, \ell]$ and every non-pad vertex w in $p_{u,v}^{(j)} \setminus p_{u,v}^{(j-1)}$ we have that $\mathsf{M}_1[(w, r)]$ can be computed using the fragments comprising the path from u to r, plus the fragment $p_{u,v}^{(j)}$. Moreover, since these fragments form paths in the SDSP tree T_r, then the path is a shortest path in G.

We now claim that each fragment contains no more than 2 times the edges contained in the true shortest path. Let $p_{u,v}$ be a subpath covering the queried shortest path. Let $p = (w_0, w_1, \ldots, w_k, v)$ in $p_{u,v}$ be the vertices precisely contained in the queried shortest path. Let 2^k be the smallest power of two at least as large as the length of p. Then there exists fragment $p_{u,v}^{(k)}$ that covers p and is at most 2 times the length of p. The for loop on line 19 of Encrypt ensures that each query is mapped to the minimum length fragment.

Storage complexity trivially follows since we pad M_1 and M_2 up to $n^2 \log n$ and $4n^2$, respectively. Computing the SDSP trees takes time $O(n^3)$ for general graphs. Each tree in $\{T_r\}_{r\in V}$ is traversed once and has a maximum size of $O(n)$. The total set of fragments for a

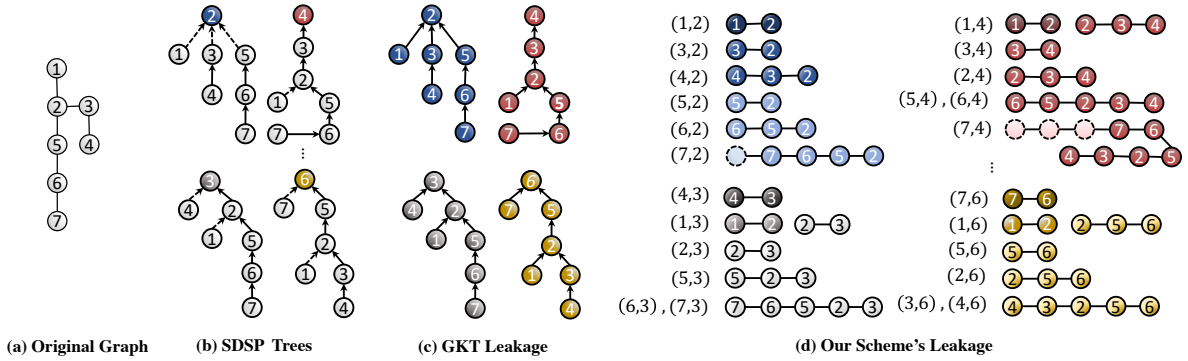


Figure 6.8: A comparison of the leakage of the GKT scheme and of our scheme. The attack in [45] against the (a) original graph results in full query recovery i.e., there exists a single isomorphism between each (b) SDSP tree and the (c) query trees computed from the GKT scheme's leakage. Thus each query can be uniquely recovered. In contrast, our scheme results in numerous fragments, with distinct queries potentially returning the same fragment. For example, queries $(6, 3)$ and $(7, 3)$ result in the same response and hence cannot be distinguished.

given SDSP tree require no more than 2 times the space of the original tree. For each tree, computing the fragments and the multimaps can be done in time linear in n. Thus the total time to encrypt is $O(n^3)$.

Issuing a query requires computing a single token using the $\mathsf{EMM\text{-}RR.Token}$. This takes $O(1)$ time and returns a single token of size $O(1)$. Any call to Search results in 1 look-up to $\mathbf{EM}_1$, $\lfloor \log n \rfloor$ look-ups to $\mathbf{EM}_2$ and the retrieval of at most $2t$ edges, for a total running time of $O(\log n + t)$. Running Reveal to decrypt the fragments is linear in the number of edges in the response, and thus takes $O(t)$ time. □

6.5 Cryptanalysis

We now describe the theoretical limitations of what an adversary can learn from our scheme's leakage. Our results demonstrate that our scheme mitigates the attack described in [45], including for families of graphs that had resulted in full query recovery.

In this section, we assume that a graph $G = (V, E)$ was encrypted using OurGES and that every possible query in $V \times V$ has been issued once. This represents the strongest passive

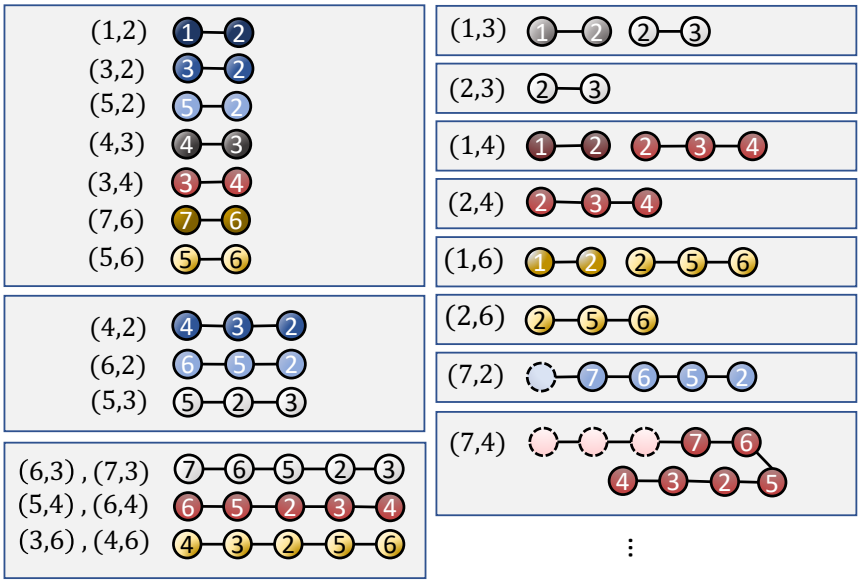


Figure 6.9: A partition of indistinguishable queries when encrypting the graph in Figure 6.8(a) with our scheme. The leakage induced by out scheme is depicted in Figure 6.8(d).

adversary without auxiliary information. Yet even in this strong setting, we demonstrate an information theoretical gap between what the adversary can reconstruct from the leakage of OurGES versus that of the GKT scheme. Furthermore, we show an exponential increase in the size of the reconstruction space for several families of graphs.

Leakage abuse attacks can broadly be categorised into ***query recovery*** attacks and ***database reconstruction*** attacks. In the context of GESs, the goal of query recovery is to infer the plaintext value of each issued query given the original graph and the query leakage. The goal of database reconstruction is to infer the underlying graph G given the setup leakage and the query leakage of a set of k queries. In either case, the adversary may attempt the attack using all possible queries or a subset of them.

The attack presented in [45] is a query recovery attack. For comparison sake, we also analyze the success of an attacker attempting query recovery against our scheme.

Definition 25 ([45])**.** *(Consistency) Let $G = (V, E)$ be a graph, $\mathcal{Q} = \{q_1, \ldots, q_k\}$ be the set of SPSP queries that are issued, and $S = \{\mathsf{tk}_1, \mathsf{tk}_2, \ldots, \mathsf{tk}_k\}$ be the corresponding set of search tokens. An assignment $\pi : S \to V \times V$ is a mapping from token sequences to SPSP*

queries. An assignment π is said to be ***consistent with the leakage*** $\mathcal{L}_{\mathsf{Q}}(G, \mathcal{Q})$ *if it satisfies* $\mathcal{L}_{\mathsf{Q}}(G, \mathcal{Q}) = \mathcal{L}_{\mathsf{Q}}(G, \pi(S))$.

In other words, an assignment π is consist with the leakage if the set of queries $\pi(S)$ could have produced the observed leakage.

Definition 26. *(QR) Let $G = (V, E)$ be a graph, $\mathcal{Q}$ be a set of SPSP queries, and S the corresponding set of search tokens. Let Π be the set of all assignments consistent with $\mathcal{L}(G, \mathcal{Q})$. The adversary achieves* ***query recovery (QR)*** *when it computes and outputs a mapping: $s \mapsto \{\pi(s) : \pi \in \Pi\}$ for all $s \in S$.*

The goal of query recovery is to compute the set of all possible SPSP queries for each search token issued by the client.

Definition 27. *(Query reconstruction space) Let $G = (V, E)$ be a graph, $\mathcal{Q}$ be a set of SPSP queries. Let Π be the set of all assignments consistent with $\mathcal{L}(G, \mathcal{Q})$. The* ***query reconstruction space*** *is the set of possible assignments, Π. The* ***size of the query reconstruction space*** *is $|\Pi|$.*

6.5.1 QR from the GKT scheme's leakage

Recall that if all possible SPSP queries are issued to the GKT scheme, then the adversary can construct from the leakage a set of n query trees; each query tree is rooted and its paths corresponds to the shortest paths whose destination is the root. These query trees are one-to-one with the SDSP trees. Thus, queries can be recovered up to the possible isomorphisms that exist between the query trees and the SDSP trees. This notion is formalized as follows.

Lemma 6.5.1 ([45])**.** *Let $G = (V, E)$ be a graph encrypted using GKT, $\{T_r\}_{r \in V}$ be the SDSP trees of G, $\mathcal{Q}$ be the set of all distinct SPSP queries, and S be the search tokens of $\mathcal{Q}$. Further let $q = (u, v)$ be a query in $\mathcal{Q}$ and $\mathsf{tk} \in S$ be its corresponding search token. If there exists a*

vertex $w \in V$ such that there is a rooted tree isomorphism $\phi : T_v \to T_w$, then there exists an assignment $\pi : S \to V \times V$ such that $\pi(\mathsf{tk}) = (\phi(u), w)$ and π is consistent with the leakage $\mathcal{L}_{\mathsf{Q}}(G, \mathcal{Q})$.

What this theorem tells us is that queries can be recovered up to the possible isomorphisms between the query trees and the SDSP trees. If there only exists one possible matching between the query trees and the SDSP trees ***and*** there only exists one isomorphism between each pair of trees, then queries can be uniquely recovered. This strong form of recovery is called ***full query recovery (FQR)*** [45]. Falzon and Paterson note that there exist families of trees for which FQR is always possible. One such family is the family of graphs that have one central vertex v and paths of distinct lengths incident to v. Figure 6.8 depicts such a graph; the left column depicts the SDSP trees and the middle column depicts the corresponding query trees.

6.5.2 QR from our scheme's leakage

In contrast to the GKT scheme, OurGES decomposes each SDSP tree into edge-disjoint paths before encrypting the paths using a response hiding EMM. As a result, an adversary cannot necessarily associate the leakage of SPSP queries with the same destination vertex whose paths are edge disjoint. Specifically, for each SDSP tree T_r, an adversary can at most recover the queries up to isomorphism of the trees rooted at the ***children*** of r.

For a concrete example, consider the graph G in Figure 6.8 and the leakage resulting from the SPSP queries $(1, 2)$, $(5, 2)$, and $(4, 3)$. The GKT scheme leaks the fact that paths $p_{1,2}$ and $p_{5,2}$ share the same destination vertex and that path $p_{4,3}$ has a different destination. In contrast, our scheme does not leak anything beyond the lengths of their respective fragments and which fragments appear together. All three queries result in a single fragment of the same size. Moreover, all three fragments correspond to only one query ad all three queries result in a response consisting of a single fragment. There is in fact no information that can

be used to distinguish these fragments or associate them as belonging to the same SDSP tree.

We characterize QR up to the isomorphisms of the subtrees rooted at the children of the SDSP tree roots as follows.

Lemma 6.5.2. *Let $G = (V, E)$ be a graph encrypted using* OurGES*, $\{T_r\}_{r \in V}$ be the SDSP trees of G, $\mathcal{Q}$ be the set of all distinct SPSP queries, and S be the search tokens of $\mathcal{Q}$. Let (u, v) be any query in $\mathcal{Q}$ and $\mathsf{tk} \in S$ be its corresponding search token. If there exists a vertex $w \in V$ and children c and d of the roots in T_v and T_w, respectively, such that there is a rooted tree isomorphism $\phi : T_v[c] \cup (c, v) \to T_w[d] \cup (d, w)$, then there exists an edge-disjoint path decomposition of $\{T_r\}_{r \in V}$ and assignment $\pi : S \to V \times V$ such that $\pi(\mathsf{tk}) = (\phi(u), w)$ and π is consistent with the leakage $\mathcal{L}_{\mathsf{Q}}(G, \mathcal{Q})$.*

Put another way, if there are two isomorphic subtrees $T_v[c]$ and $T_w[d]$, then there is a set of queries of the form (a, b) where $a \in T_v[c]$ that is indistinguishable from a set of queries (a', b') where $a' \in T_w[d]$.

Proof. We first define π. Let $A \subseteq \mathcal{Q}$ be the set of queries (a, b) such that $a \in T_v[c]$ and $b = v$, and let $S_A \subseteq S$ be the tokens generated by A. Similarly, let $B \subseteq \mathcal{Q}$ be the set of queries (a, b) such that $a \in T_w[d]$ and $b = w$, and let $S_B \subseteq S$ be the tokens generated by B. Let $\mathsf{tk} \in S$ and (a, b) be the query that generated tk:

- If $\mathsf{tk} \in S_A$, then set $\pi(\mathsf{tk}) = (\phi(a), \phi(b))$.
- If $\mathsf{tk} \in S_B$, then set $\pi(\mathsf{tk}) = (\phi^{-1}(a), \phi^{-1}(b))$.
- Else set $\pi(\mathsf{tk}) = (a, b)$.

We now show that π is consistent with the leakage i.e., that the query and structure pattern of $\mathcal{L}_{\mathsf{Q}}(G, \mathcal{Q})$ and $\mathcal{L}_{\mathsf{Q}}(G, \pi(S))$ are equal. Since $\mathcal{Q}$ is a set and π is a permutation, then $\pi(S)$ is also a set. Thus the query pattern for both is a $|\mathcal{Q}| \times |\mathcal{Q}|$ matrix with 1's along the diagonal and 0's everywhere else.

Let $H = (\mathcal{Q}, F)$ and $H' = (\pi(S), F')$ be the structure pattern graphs of $\mathcal{L}_{\mathsf{Q}}(G, \mathcal{Q})$ and $\mathcal{L}_{\mathsf{Q}}(G, \pi(S))$, respectively. It remains to show that the structure pattern is equal, i.e. that there is an edge-weight preserving isomorphism $\varphi : H \to H'$. Observe that the subgraph of H induced by A and its neighbors is disconnected from the lest of the graph. Similarly for B. This is because all shortest paths in $T_v[c] \cup (c, v)$ are edge disjoint from all other shortest paths and so does not share fragments with any other queries.

We now define φ for each of these components of H. Now consider the queries $q \in A$. Any edge-disjoint decomposition of T_v must be a decomposition for $T_v[c]$. Since the rooted-tree isomorphism ϕ is edge-preserving, then any edge-disjoint decomposition of $T_v[c]$ must also be a decomposition for $T_w[d]$ under ϕ. Moreover, since ϕ is a map of vertices to vertices, it also maps shortest paths to shortest paths and fragments of a particular length to fragments of the same length. We abuse notation of $\phi(f)$ to mean the image of the vertices in fragment f under ϕ.

- For any edge $(q = (a, b), f) \in \mathcal{Q} \times F$ such that $q \in A$, set $\varphi((a, b)) = (\phi^{-1}(a), \phi^{-1}(b))$.
- For any edge $(q = (a, b), f) \in \mathcal{Q} \times F$ such that $q \in B$, set $\varphi((a, b)) = (\phi^{-1}(a), \phi^{-1}(b))$.
- For any edge $(q, f) \in \mathcal{Q} \times F$ such that $\mathcal{Q} \setminus (A \cup B)$, then $q \in \pi(S)$ and $f \in F'$. For all such vertices, set φ as the identity, i.e. $\varphi(q) = q$ and $\varphi(f) = f$.

Since φ maps fragments under the identity or ϕ, then it is edge-weight preserving (the edge weights of H and H' correspond to the lengths of the incident fragments). To see that φ is also edge-preserving, let $(q = (a, b), f)$ be any edge in H. If $q \in A$ or $q \in B$, then by isomorphism of ϕ either $((\phi(a), \phi(b)), \phi(f)) \in H'$ or $((\phi^{-1}(a), \phi^{-1}(b)), \phi^{-1}(f)) \in H'$, respectively. Else $q \in \mathcal{Q} \setminus (A \cup B)$. Since $\pi^{-1}(q)$ is the token generated by q, then $(\varphi(q), \varphi(f)) \in H'$. Thus $H \cong H'$ and π is consistent with $\mathcal{L}_{\mathsf{Q}}(G, \mathcal{Q})$. □

6.5.3 Reconstruction Space

We now make some additional and more general observations about the reconstruction space of our scheme.

Lemma 6.5.3. *Let $G = (V, E)$, and fix a set of SDSP trees for G and a decomposition of each tree into canonical fragments. For each canonical fragmentof length L, there are at least $L/2$ equivalent queries.*

Proof. Let $v \in V$ and let f be a fragment in SDSP tree T_v. Let the corresponding path of f in T_v be $(u_0, u_2, \ldots, u_{k^2})$ where u_{k^2} is the node closest to the root v. Then set of queries $\{(u_i, v) : 0 \leq i \leq k^2/2\}$ all result in the same response. □

Corollary 6.5.4. *Let $\mathcal{F}$ be the set of all fragments. Then the reconstruction space is of size at least $\prod_{f \in \mathcal{F}, |f| \geq 2} \frac{|f|}{2}!$*

For concreteness, we compute the sizes of the reconstruction spaces of the GKT scheme versus our scheme for various graph families. The results can be found in Table 6.1 and the proofs of these bounds can be found below.

6.5.4 Reconstruction Space Lowerbounds

Complete Graph

Lemma 6.5.5. *Let $G = (V, E)$ be a complete graph on n vertices and let $\mathcal{Q} = V \times V$. Let G be encrypted using the GKT scheme and let $\mathcal{L}(G, \mathcal{Q})$ be its corresponding leakage. The size of the reconstruction space is $(n!)^{n+1}$.*

Proof. The SDSP trees of a complete graph are n stars, each star rooted at a different root node of G. Each star is isomorphic. It follows that there are n query trees, all of which are

isomorphic stars. There are $n!$ ways to assign the query trees to the SDSP trees. For each pair, there are $n!$ isomorphisms. Since there are n such pairs we have a total reconstruction space size of $n! \cdot (n!)^n$. □

Lemma 6.5.6. *Let $G = (V, E)$ be a complete graph on n vertices and let $\mathcal{Q} = V \times V$. Let G be decomposed into non and encrypted using our scheme, and let $\mathcal{L}(G, \mathcal{Q})$ be its corresponding leakage. The size of the reconstruction space is $(n^2)!$.*

Proof. The SDSP trees of a complete graph are n stars. Decomposing each tree into disjoint paths results in n paths each of length 1. In total there are n^2 fragments all of which are indistinguishable. There are $(n^2)!$ possible bijections between $V \times V$ and the set of n^2 fragments and the lemma follows. □

Line Graph

Lemma 6.5.7. *Let $G = (V, E)$ be a path on n vertices and $\mathcal{Q} = V \times V$. Let G be encrypted using the GKT scheme and let $\mathcal{L}(G, \mathcal{Q})$ be its corresponding leakage. The size of the reconstruction space is $2^{n/2}$.*

Proof. WLOG, let the graph be the path $(1, 2, 3 \ldots, n)$ and let n be even. Graph G has n corresponding SDSP trees. Observe that the SDSP tree rooted at node $i \in V$ is isomorphic to the SDSP tree rooted at $n - i + 1$. Given the set of n query trees, each query tree is isomorphic to exactly two SDSP trees: the SDSP tree it actually corresponds to and its reflection. For each SDSP tree and query tree pair that are isomorphic, there exists only one isomorphism between the pair of trees. This is because each tree has at most two paths, each of differing lengths, descending from the root, and thus the isomorphism is determined. The reconstruction space is thus the number of ways that to pair the SDSP trees with isomorhic query trees, multiplied by the number of isomorphisms between each pair. The size of the reconstruction space is $2^{n/2}$. A similar argument holds for odd n. □

Lemma 6.5.8. *Let $G = (V, E)$ be a line graph on n vertices and let $\mathcal{Q} = V \times V$. Let G be decomposed into non and encrypted using our scheme, and let $\mathcal{L}(G, \mathcal{Q})$ be its corresponding leakage. The size of the reconstruction space is $\Omega((n!)^{\log n - 1})$.*

Proof. WLOG, let the graph be the path $(1, 2, 3 \ldots, n)$. Each SDSP tree is comprises of two paths descending from the root. When the SDSP tree is decomposed, each path consequently becomes a disjoint path. Any query thus results in a response comprised of a single canonical fragment. Thus, each canonical fragment of the same length and which does not comprise of any padding vertices are indistinguishable. To obtain this lower bound we will count the number of canonical fragments of the same length. There are at least n canonical fragments of each length $1, 2, 2^2, \ldots, 2^{\log n - 1}$. Each canonical fragment of length ℓ length is indistinguishable from any other fragment of the same length. This gives us a lowerbound of $(n!)^{\log n - 1}$. □

Asymmetric Star

Lemma 6.5.9. *Let $G = (V, E)$ be an asymmetric star graph with one central vertex and paths of distinct length $k_1, \ldots, k_\ell$ attached to the central vertex, and let $\mathcal{Q} = V \times V$. Let G be encrypted using the GKT scheme and let $\mathcal{L}(G, \mathcal{Q})$ be its corresponding leakage. The size of the reconstruction space is* 1.

This result follows from Section 4.9 of [45].

Lemma 6.5.10. *Let $G = (V, E)$ be an asymmetric star graph with one central vertex and paths of distinct length $k_1, \ldots, k_\ell$ attached to the central vertex, and let and let $\mathcal{Q} = V \times V$. Let G be decomposed into non and encrypted using our scheme, and let $\mathcal{L}(G, \mathcal{Q})$ be its corresponding leakage. The size of the reconstruction space is $\Omega\left(\prod_{i=1}^{\ell}\left(\prod_{j=1, j\neq i}^{\ell} \frac{k_i}{2}!\right)^{k_i}\right)$.*

Proof. Graph G is a graph with one central vertex and paths of distinct length $k_1, \ldots, k_\ell$ attached to the central vertex. Fix $i \in [\ell]$ and consider an SDSP tree rooted at any vertex

Dataset	Graph Characteristics						Setup						
	$\|V\|$	$\|E\|$	d	Num of Comp	Dia-meter	Max Frag Length	M_1 Size	M_1 % Pad	$\mathbf{EM}_1$ Size	M_2 Size	M_2 % Pad	$\mathbf{EM}_2$ Size	Total Time
InternetRouting	35	323	0.543	1	2	2	202KB	87.3	269KB	591KB	30.4	619KB	360ms
CA-GrQc	46	1030	0.995	1	2	2	351KB	85.3	468KB	1039KB	30.2	1076KB	442ms
email-Eu-core	1005	16,706	0.0331	20	7	4	238MB	64.8	293MB	505MB	29.8	525MB	41s
facebook-combined	4039	88,234	0.011	1	8	4	4.39GB	60.7	5.25GB	8.46GB	30.2	8.48GB	11.1min
p2p-Gnutella08	6301	20,777	0.001	2	9	8	11.3GB	61.0	13.4GB	19.9GB	31.5	20.6GB	26.6min
p2p-Gnutella04	10,876	39,994	0.0006	1	10	8	35.7GB	58.3	41.9GB	59.5GB	31.6	61.5GB	1.47hr
p2p-Gnutella25	22,687	54,705	0.0002	13	11	8	164GB	57.5	190GB	265GB	31	268GB	7.39hr
Swiss	19,976	24,009	0.00012	1	311	512	127GB	54.9	148GB	127GB	34.8	207GB	5.88hr
Cali	21,693	21,693	0.00009	2	491	512	150GB	53.6	174GB	119GB	32.7	245GB	8.51hr

Table 6.2: Details about the real-world datasets used in our experiments and our setup experimental results. $|V|$ denotes the number of vertices, $|E|$ the number of edges of the graph dataset, and $d = 2|E|/(|V|^2 - |V|)$ the density of the graph.

along the path of length k_i. This SDSP tree comprises of some descendenent of the root to which the other paths of lengths $\{k_j\}_{j\neq i, j\in[\ell]}$ are incident. Applying Corollary 6.5.4 to these paths, each path of length k_j gives rise to at least $k_j/2$ indistinguishable queries. These queries are indistinguishable and hence can be permuted in any way possible i.e. $\left(\prod_{j=1,j\neq i}^{\ell} \frac{k_j}{2}!\right)$. There are k_i such SDSP trees for any fixed i and ℓ different values for i. This yields a lower bound of $\prod_{i=1}^{\ell} \left(\prod_{j=1,j\neq i}^{\ell} \frac{k_j}{2}!\right)^{k_i}$. □

6.6 Empirical Evaluation

We now evaluate our scheme's performance on real world graph datasets and compare its performance to the GKT scheme [52].

Experimental setup. We implemented our scheme using Python 3.7.6 and ran our experiments on a compute cluster with a 32 Core Intel Xeon (Cascade Lake-based) processor and 2146GB Memory. For comparison, we also implemented the GKT scheme. Both implementations used the same compute node for the client and the server, so our results do

not include network latency.

Graphs. We used the `NetworkX` library version 2.6.3 [98] to represent and manipulate graphs. We used our own implementation of the heavy-light decomposition algorithm (Figure 2).

Cryptographic primitives. The cryptographic primitives were implemented using the `Cryptography` library version 39.0.1 [108]. For symmetric encryption we used AES in CBC mode with 16B block size and key length; for cryptographic hash functions we used SHA-256; for search token generation we used HMAC with SHA-256. We implemented our EMM schemes using [23] and encryption was parallelized across 20 cores.

6.6.1 Datasets

We evaluated our scheme on the same social network datasets as Ghosh et al. [52] and Falzon and Paterson [45], along with two geographical datasets: the Swiss Federal Railway timetable [110] and the California road network [79]. See Table 6.2 for more details.

- **InternetRouting [77]:** A dataset from the University of Oregon Route Views Project. A dense subgraph ($n = 35$) was extracted using the dense subset extraction algorithm by Charikar [28] as implemented by Ambavi et al. [6].

- **Ca-GrQc [77]:** A network of the General Relativity and Quantum Cosmology arXiv collaborations from January 1993 to April 2003. A subset ($n = 46$) was extracted using dense subset extraction.

- **email-EU-core [77]:** A network of internal emails sent between members of a large European research institution. We parsed the data as a non-directed graph, i.e. an edge (u, v) exists if either u sent v and email or vice versa.

- **facebook-combined [77]:** A social network derived from Facebook friends lists. The graph includes all edges from the ego networks collected in the survey.

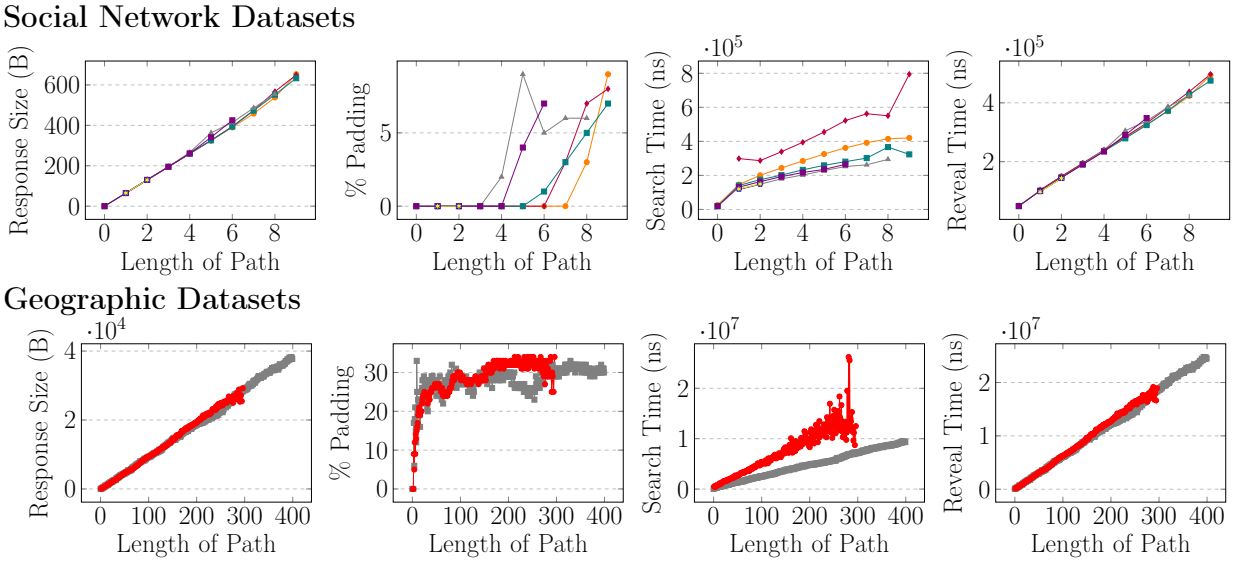


Figure 6.10: Query benchmarks with respect to the length of the path queried. We use the following symbols for the social network datasets: InternetRouting (—✳—), Ca-GrQc (—●—), email-EU-core (—■—), facebook-combined (—▲—), p2p-Gnutella08 (—■—), p2p-Gnutella04 (—◆—), p2p-Gnutella25 (—●—). And the following symbols for the geographic datasets: Swiss (—●—) and Cali (—■—). For each dataset we issued 100,000 random queries, partitioned them based on path length, and took the average of the respective attribute within each set of the partition.

- **p2p-Gnutella [77]:** Four datatsets depicting the Gnutella peer-to-peer network from the 4th, 5th, and 25th of August 2002.

- **Swiss [110]:** A timetable of the Swiss Railway from December 13, 2015 to December 10, 2016 parsed as a graph [85]. This graph comprised of 178 components as a result of numerous funiculars and short connections that are all a part of the network. We extracted the largest component from this graph.

- **Cali [79]:** A dataset of the California Road Network. This dataset was also used in prior works on range schemes e.g. [44, 87].

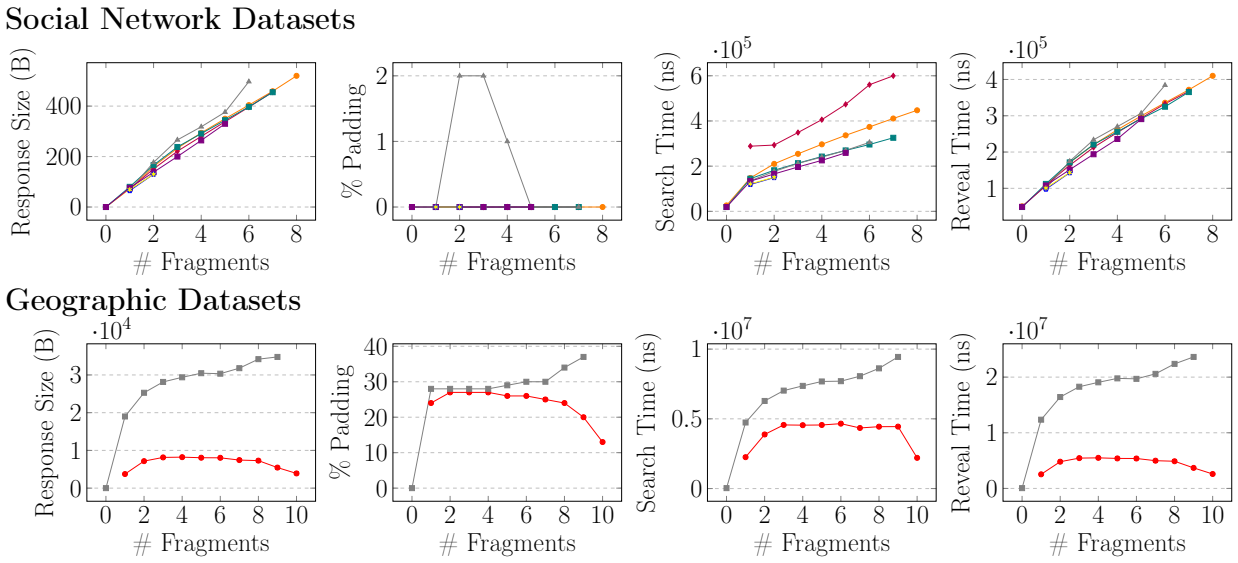


Figure 6.11: Response size, Search time at server, and Reveal time at client as they vary with (Top row) the length of the path queried and (Bottom row) the number of fragments returned. We use the following symbols for the social network datasets: InternetRouting (—✳—), Ca-GrQc (—●—), email-EU-core (—■—), facebook-combined (—▲—), p2p-Gnutella08 (—■—), p2p-Gnutella04 (—◆—), p2p-Gnutella25 (—●—). And the following symbols for the geographic datasets: Swiss (—●—) and Cali (—■—). For each dataset we issued 10,000 random queries, partitioned them based on path length (or number of fragments), and took the average of the respective attribute within each set of the partition.

6.6.2 Performance

Setup. Setup includes the KeyGen and Encrypt algorithms. We report setup benchmarks in Table 6.2, including sizes of the plaintext multimaps, percent padding, sizes of the encrypted multimaps, and total setup time. Setup times were practical ranging from as little as 360 ms ($n = 35$) to 8.51 hours ($n = 21,693$).

Client-side storage required only a 32B key, one 16B key each for the response-hiding EMM and the response-revealing EMM. Similar to the GKT scheme, server-side storage is the most significant cost of our scheme. Total storage ranged from 888KB ($n = 35$) to 458GB ($n = 22,687$). Due to the way padding was added, the sizes of $\mathbf{EM}_1$ and $\mathbf{EM}_2$ were at most 1.33 times and 2.06 times larger than the plaintext multimaps, respectively. Storage is proportional to the number of nodes and is independent of the density of the graph. In

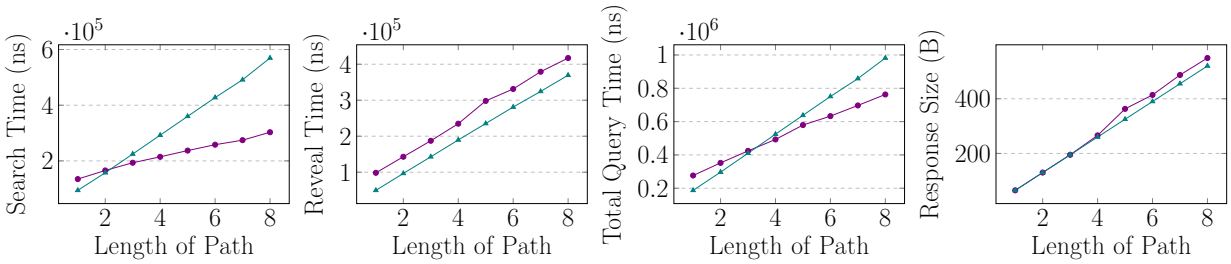


Figure 6.12: OurGES (—●—) and GKT (—▲—) query benchmarks for the facebook-combined dataset with respect to the length of the queried path. Results were averaged over 100,000 uniformly random issued SPSP queries. Observe that Search takes almost half an order of magnitude more for GKT, whereas Reveal is faster for GKT. Despite the worst-case 2x overhead in bandwidth of OurGES, the response size is on average comparable to the bandwidth of GKT.

contrast, the GKT scheme's encryption overhead varies with the density of the graph.

Query. Querying the database comprises of three algorithms: Token (client-side computation of the search token), Search (server-side look-up of the encrypted records), and Reveal (client-side decryption of the response). In Figures 6.10 and 6.11 we depict query benchmarks for all datasets including the response size, percent padding of the response, runtime of Search, and runtime of Reveal. For each dataset we sampled 100,000 uniformly random queries, partitioned the queries based on length of the queried path (or respectively, the number of fragments), and averaged the benchmarks within each set of the partition.

In general, response size varies proportionally with the length of the path, with an approximate 100B increase for each additional vertex in the path. We found that experimentally Search increases linearly with respect to both the length of the path and the number of fragments. This is due to the fact that longer paths are also likely to result in more fragments. Similarly, Reveal increases linearly with respect to both number of fragments and length of the path. However, we see a closer correlation to length since response decryption is only dependent on the number of edges returned.

6.6.3 Comparison with GKT

Figure 6.12 depicts query benchmarks for the facebook-combined dataset for both OurGES and GKT. We issued 100,000 uniformly random SPSP queries, partitioned the results with respect to the length of the shortest path returned, and averaged the respective attribute for all shortest paths of a given length.

Token is very efficient for both schemes and takes approximately 0.043ms. Search runtime is faster for our scheme despite what the theoretical complexity might suggest. This discrepancy between worst-case asymptotic complexity and real-world performance can be largely attributed to the difference in number of calls to cryptographic primitives that our implementations of the response-revealing and response-hiding EMM schemes must make. In particular, a call to Search.EMM-RR requires computing for each value: one hash evaluation (to obtain the encrypted label) and one symmetric decryption (to reveal the value). In contrast, a call to Search.EMM-RH requires computing for each value only one hash evaluation (to obtain the encrypted label). The GKT scheme looks up t edges in a dictionary encrypted using the response-revealing scheme. Our scheme instead looks up the (at most) $t + \delta$ edges in $\mathbf{EM}_2$ for $\delta \in [1, t + 1]$, which are encrypted using the response-hiding scheme. Note that in practice δ is closer to 1.

Note that our scheme's search time could further be decreased through parallelization since the look up for each fragment can be processed using a different core. In contrast, the GKT scheme's search cannot be parallelized since the vertices must be looked up sequentially.

Reveal is marginally faster for the GKT scheme, since it only involves symmetric decryption of the vertices in the shortest path. In our scheme, the returned fragments may include additional nodes and padding vertices that are not in the shortest path. Moreover, the Reveal algorithm of our response-hiding EMM implementation also entailed a key-derivation step, hence the small additive increase in decryption time required by OurGES.

Total query time (which includes runtime of Token, Search, and Reveal) is efficient and

practical for both schemes. However, as the length of the path increases, the total query time of GKT overtakes that of OurGES. Response size increases linearly with respect to the path length for both schemes. On average, the response size of OurGES almost matches that of GKT, despite the worst-case 2x constant factor overhead in bandwidth for our scheme.

6.7 Conclusion

In this work, we present a new graph encryption scheme for shortest path queries. We describe a new data structure for storing shortest and responding to SPSP queries. Our scheme is built upon this data structure and achieves optimal bandwidth complexity while mitigating the attack by Falzon and Paterson [45]. We support our scheme with a proof of security and thorough cryptanalysis. We experimental evaluated our scheme to demonstrate its practicality on a number of real-world social network and geographic datasets. We leave open the problem of designing schemes for supporting other important queries on graphs, such as k-nearest neighbors.

CHAPTER 7

CONCLUSION

In this thesis we have studied structured encryption schemes that support complex expressive queries – i.e., ***range queries on multi-attribute databases*** (Chapters 3 and 4) and ***single-pair shortest path queries on graphs*** (Chapters 5 and 6) – through both an ***adversarial lens*** (Chapters 3, 4, and 5) and a ***constructive one*** (Chapter 6). We applied a number of combinatorial and algorithmic techniques to our work, and extended them to the setting of encrypted databases to obtain new results that may be of independent interest.

7.1 Why these Attacks Matter

One of the most common criticisms of existing attacks is that the assumptions are too strong and too unrealisic, pointing to papers that require every query to be issued e.g., [43, 70] or require knowledge of the query distribution [58, 72, 75]. Despite the low likelihood that a client will issue all queries, we cannot dismiss the fact that doing so has been shown detrimental to security time and time again. In many other areas of cryptography, the existence of even one polynomial-time attack against a scheme or primitive would call into question its security. Why should structured encryption be any different?

Given that passive adversaries can recover so much information – and in polynomial time no less – suggests that these schemes are leaking too much information to begin with and that we should instead focus on building schemes that are secure *regardless* of the number of queries issued or the query distribution from which the queries are drawn. The attacks that we have described in this thesis should be viewed as a reminder to tread with caution and re-evaluate how we build these schemes and evaluate their security.

7.2 The Importance of Cryptanalysis

Yehuda Lindell wrote: "*There are plenty of [cryptographers] who dabble in theory and practice, or may be somewhere on the spectrum in the middle. Despite their differences, I personally think that they have a lot in common. The adversarial mindset that cryptographers need to do their job is the same whatever they are doing*" [81]. This quote emphasizes that even when we are constructing a scheme, we should be thinking like an adversary. That often means assuming the worst-case scenario and not dismissing the possibility that some hospital staff member may, at some point, accidentally request all patients in the system between the ages of 0 and 120.

In Chapter 6, our cryptanalysis was done in conjunction with security proofs. Not only do we establish the leakage profile of our scheme, but we also prove information theoretic results about what a passive adversary cannot learn. It is our hope that future schemes are also accompanied by cryptanlysis and theorems that bound the type of information that adversaries can learn. Such cryptanlysis will serve as a forcing function for the community to better understand the schemes that we put forth. Moreover, it will help us overcome the cycle of building schemes, attacking them, and then building a new modified scheme that can ultimately be similarly attacked.

7.3 Directions for Future Work

Structured encryption is a promising approach for private queries on outsourced data. That said, there is still much work to be done on developing structured encryption that is efficient, sufficiently secure, and supports expressive queries.

Kornaropoulos et al. [71] recently proposed a quantitative measure of searchable encryption via ***leakage inversion***. Their key insight is that leakage is a function and that the preimage of a leakage pattern is the set of equivalent databases that could have produced that leakage.

One could thus apply standard information theoretic techniques to compute, for example, the ***entropy of the reconstruction space*** and better compare different leakage profiles. They define closed-form expressions and derive bounds for both keyword-based and range-based databases and one interesting direction would be to extend their analysis to other types queries. It would also be interesting to explore whether other information theoretic techniques would be helpful in quantifying leakage.

On the constructive side, I am interested in designing graph encryption schemes with smaller storage overhead than that presented in Chapter 6. More generally, the question of how to design a scheme that supports a number of standard graph queries (e.g., k-nearest node queries, single-pair shortest path queries, diameter queries) remains open. In order to make schemes practical and move towards real-world deployment, we must ultimately design and analyze schemes that are functional, expressive, efficient, and dynamic.

REFERENCES

[1] Agency for Healthcare Research and Quality. Healthcare Cost and Utilization Project (HCUP). Nationwide Inpatient Sample (NIS) datasets NIS 2008 and 2009, `https://www.hcup-us.ahrq.gov/`, 2008, 2009.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD 2004, 2004.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey Ullman. *Data Structures and Algorithms*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1983.

[4] Akshima, David Cash, Francesca Falzon, Adam Rivkin, and Jesse Stern. Multidimensional database reconstruction from range query access patterns. Cryptology ePrint Archive, Report 2020/296, 2020. `https://eprint.iacr.org/2020/296`.

[5] Amazon. Amazon neptune, 2021. Accessed on October 27, 2021.

[6] Heer Ambavi, Mridul Sharma, and Varun Gohil. Densest-subgraph-discovery. `https://github.com/varungohil/Densest-Subgraph-Discovery`, 2020.

[7] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *Proc. Priv. Enhancing Technol.*, 2023(1):417–436, 2023.

[8] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13. ACM New York, NY, USA, 2013.

[9] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. The tao of inference in privacy-protected databases. *Proc. VLDB Endow.*, 11(11):1715–1728, July 2018.

[10] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. The guard's dilemma: Efficient Code-Reuse attacks against intel SGX. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1213–1227, Baltimore, MD, aug 2018. USENIX Association.

[11] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2009*, 2009.

[12] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Advances in Cryptology – CRYPTO 2011*, 2011.

[13] R. Bost. Sophos: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1143–1154, New York, NY, USA, 2016. Association for Computing Machinery.

[14] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[15] 7 most infamous cloud security breaches. accessed on February 25, 2022.

[16] Ernie Brickell, Gary Graunke, Michael Neve, and Jean-Pierre Seifert. Software mitigations to hedge aes against cache-based software side channel vulnerabilities, 2006.

[17] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, and Venkat Venkataramani. TAO: Facebook's distributed data store for the social graph. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 49–60, San Jose, CA, June 2013. USENIX Association.

[18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, page 991–1008, Baltimore, MD, aug 2018. USENIX Association.

[19] K. P. Burnham and W. S. Overton. Estimation of the size of a closed population when capture probabilities vary among animals. *Biometrika*, 65(3):625–633, 1978.

[20] K. P. Burnham and W. S. Overton. Robust estimation of population size when capture probabilities vary among animals. *Ecology*, 60(5):927–936, 1979.

[21] Alina Campan, Yasmeen Alufaisan, and Traian Marius Truta. Preserving communities in anonymized social networks. *Trans. Data Privacy*, 8(1):55–87, dec 2015.

[22] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proc. ACM Conf. on Computer and Communications Security*, CCS, 2015.

[23] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In *21st Annual Network and Distributed System Security Symposium 2014*, NDSS 2014, 2014.

[24] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology*, CRYPTO, 2013.

[25] David Cash, Ruth Ng, and Adam Rivkin. Improved structured encryption for SQL databases via hybrid indexing. In *Applied Cryptography and Network Security, Proceedings, Part II*, volume 12727 of *LNCS*, pages 480–510. Springer, 2021.

[26] Edwin Chan. Pydistinct, 2020.

[27] Anne Chao and Shen-Ming Lee. Estimating the number of classes via sample coverage. *Journal of the American Statistical Association*, 87(417):210–217, 1992.

[28] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In Klaus Jansen and Samir Khuller, editors, *Approximation Algorithms for Combinatorial Optimization*, pages 84–95, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[29] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology – ASIACRYPT 2010*, 2010.

[30] Ciphercloud. Ciphercloud: Cloud data security company, 2021. Accessed on April 26, 2021.

[31] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, page 937–946, USA, 2002. Society for Industrial and Applied Mathematics.

[32] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[33] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 79–88, New York, NY, USA, 2006. Association for Computing Machinery.

[34] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[35] Marc Damie, Florian Hahn, and Andreas Peter. A highly accurate query-recovery attack against searchable encryption using non-indexed documents. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 143–160. USENIX Association, 2021.

[36] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage. In *27th Annual Network and Distributed System Security Symposium 2020*, NDSS 2020, 2020.

[37] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. Practical private range search revisited. In *Proc. ACM Int. Conf. on Management of Data*, SIGMOD, 2016.

[38] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, Minos Garofalakis, and Charalampos Papamanthou. Practical private range search in depth. *ACM Trans. Database Syst.*, 43(1), 2018.

[39] Minxin Du, Peipei Jiang, Qian Wang, Sherman S. M. Chow, and Lingchen Zhao. Shielding graph for exact analytics with sgx. *IEEE Transactions on Dependable and Secure Computing*, pages 1–11, jan 2023.

[40] F. Betül Durak, Thomas M. DuBuisson, and David Cash. What else is revealed by order-revealing encryption? In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1155–1166, New York, NY, USA, 2016. Association for Computing Machinery.

[41] Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983.

[42] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In *20th European Symposium on Research in Computer Security 2015*, ESORICS 2015, 2015.

[43] Francesca Falzon, Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia. Full Database Reconstruction in Two Dimensions. In *Proc. ACM Conf. on Computer and Communications Security*, CCS, 2020.

[44] Francesca Falzon, Evangelia Anna Markatou, Zachary Espiritu, and Roberto Tamassia. Range search over encrypted multi-attribute data. volume 16(4), pages 587–600, 2022.

[45] Francesca Falzon and Kenneth G. Paterson. An efficient query recovery attack against a graph encryption scheme. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part I*, volume 13554 of *Lecture Notes in Computer Science*, pages 325–345. Springer, 2022.

[46] Stefan Felsner and Lorenz Wernisch. Maximum k-chains in planar point sets: Combinatorial structure and algorithms. *SIAM J. Comput.*, 28(1):192–209, 1998.

[47] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

[48] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John D. Mitchell, and Robert K. Cunningham. SoK: Cryptographically protected database search. In *Proc. IEEE Symposium on Security and Privacy 2017*, S&P 2017, 2017.

[49] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *Advances in Cryptology - CRYPTO 2016*, 2016.

[50] Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*, volume 20:09. Stanford university Stanford, 2009.

[51] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1038–1055, New York, NY, USA, 2018. Association for Computing Machinery.

[52] Esha Ghosh, Seny Kamara, and Roberto Tamassia. Efficient graph encryption scheme for shortest path queries. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, page 516–525, New York, NY, USA, 2021. Association for Computing Machinery.

[53] Anselme Goetschmann. *Design and Analysis of Graph Encryption Schemes*. Master's thesis, ETH Zürich, 2020.

[54] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 182–194, New York, NY, USA, 1987. Association for Computing Machinery.

[55] Oded Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *Proc. ACM Symp. on Theory of Computing*, STOC, 1987.

[56] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *Proc. IEEE Symp. on Security and Privacy 2017*, S&P 2017, 2017.

[57] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. Pancake: Frequency smoothing for encrypted data stores. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[58] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 315–331. ACM, 2018.

[59] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *Proc. IEEE Symp. on Security and Privacy 2019*, S&P 2019, 2019.

[60] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. Encrypted databases: New volume attacks against range queries. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 361–378. ACM, 2019.

[61] P. Haas, J. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *VLDB*, 1995.

[62] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[63] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, New York, NY, USA, 2013. Association for Computing Machinery.

[64] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 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, 2012.

[65] Seny Kamara and Tarik Moataz. Sql on structurally-encrypted databases. In *Advances in Cryptology – ASIACRYPT 2018*, 2018.

[66] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In *Advances in Cryptology – EUROCRYPT 2019*, 2019.

[67] Seny Kamara, Tarik Moataz, Stan Zdonik, and Zheguang Zhao. An optimal relational database encryption scheme. Cryptology ePrint Archive, Report 2020/274, 2020. `https://eprint.iacr.org/2020/274`.

[68] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security*, pages 258–274, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[69] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proc. ACM Conf. on Computer and Communications Security*, CCS. ACM, 2012.

[70] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *Proc. ACM Conf. on Computer and Communications Security 2016*, CCS 2016, 2016.

[71] Evgenios M. Kornaropoulos, Nathaniel Moyer, Charalampos Papamanthou, and Alexandros Psomas. Leakage inversion: Towards quantifying privacy in searchable encryption. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1829–1842, New York, NY, USA, 2022. Association for Computing Machinery.

[72] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Data recovery on encrypted databases with k-nearest neighbor query leakage. In *Proc. IEEE Symp. on Security and Privacy 2019*, S&P 2019, 2019.

[73] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. In *Proc. IEEE Symp.on Security and Privacy 2020*, S&P 2020, 2020.

[74] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Response-hiding encrypted ranges: Revisiting security via parametrized leakage-abuse attacks. In *Proc. IEEE Symp. on Security and Privacy*, S&P, 2021.

[75] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *Proc. IEEE Symp. on Security and Privacy 2018*, S&P 2018, 2018.

[76] Russell W. F. Lai and Sherman S. M. Chow. Forward-secure searchable encryption on labeled bipartite graphs. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *Applied Cryptography and Network Security*, pages 478–497, Cham, 2017. Springer International Publishing.

[77] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[78] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In Claudia Bauzer Medeiros, Max J. Egenhofer, and Elisa Bertino, editors, *Advances in Spatial and Temporal Databases*, pages 273–290, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[79] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In Claudia Bauzer Medeiros, Max J. Egenhofer, and Elisa Bertino, editors, *Advances in Spatial and Temporal Databases*, pages 273–290, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[80] Mingyue Li, Chunfu Jia, Ruizhong Du, and Wei Shao. Forward and backward secure searchable encryption scheme supporting conjunctive queries over bipartite graphs. *IEEE Transactions on Cloud Computing*, pages 1–1, 2021.

[81] Yehuda Lindell. What do cryptographers do? https://www.linkedin.com/pulse/what-do-cryptographers-yehuda-lindell/, feb 2020.

[82] Chang Liu, Liehuang Zhu, and Jinjun Chen. Graph encryption for top-k nearest keyword search queries on cloud. *IEEE Transactions on Sustainable Computing*, 2(4):371–381, 2017.

[83] Chang Liu, Liehuang Zhu, Xiangjian He, and Jinjun Chen. Enabling privacy-preserving shortest distance queries on encrypted graph data. *IEEE Trans. Dependable Secur. Comput.*, 18(1):192–204, jan 2021.

[84] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, April 2012.

[85] Kevin Scott Mader. Parsing sbb routes as a graph. https://www.kaggle.com/code/kmader/parsing-sbb-routes-as-a-graph/notebook, 2019.

[86] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, page 135–146, 2010.

[87] Evangelia Anna Markatou, Francesca Falzon, Roberto Tamassia, and William Schor. Reconstructing with less: Leakage abuse attacks in two-dimensions. In *Proc. ACM Conf. on Computer and Communications Security*, CCS, 2021.

[88] Evangelia Anna Markatou and Roberto Tamassia. Full database reconstruction with access and search pattern leakage. In *Proc. Int. Conf on Information Security 2019*, ISC 2019, 2019.

[89] Evangelia Anna Markatou and Roberto Tamassia. Mitigation techniques for attacks on 1-dimensional databases that support range queries. In *Information Security - 22nd International Conference, ISC 2019*, 2019.

[90] Evangelia Anna Markatou and Roberto Tamassia. Database reconstruction attacks in two dimensions. Cryptology ePrint Archive, Report 2020/284, 2020. https://eprint.iacr.org/2020/284.

[91] McAfee. Mcafee, 2021. accessed on April 26, 2021.

[92] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, New York, NY, USA, 2013. Association for Computing Machinery.

[93] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *Proc. ACM Conf. on Computer and Communications Security*, CCS '15, page 504–517, 2015.

[94] Kyriakos Mouratidis and Man Lung Yiu. Shortest path computation with no information leakage. *Proceedings of the VLDB Endowment*, 5(8):692–703, 2012.

[95] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *Proc. ACM Conf. on Computer and Communications Security 2015*, CCS 2015, 2015.

[96] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*, pages 639–654, 2014.

[97] Inc. Neo4j. Neo4j, 2021. Accessed on October 27, 2021.

[98] NetworkX, 2021. version 2.6.2.

[99] Ontotext. GraphDB, 2021. Accessed on October 27, 2021.

[100] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, August 2021.

[101] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. Big data analytics over encrypted datasets with seabed. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 587–602, Savannah, GA, nov 2016. USENIX Association.

[102] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 79–93, 2019.

[103] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. Arx: An encrypted database using semantically secure encryption. *Proc. VLDB Endow.*, 12(11):1664–1678, August 2019.

[104] Rishabh Poddar, Stephanie Wang, Jianan Lu, and Raluca Ada Popa. Practical volume-based attacks on encrypted databases. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 354–369. IEEE, 2020.

[105] Geong Sen Poh, Moesfa Soeheila Mohamad, and Muhammad Reza Z'aba. Structured encryption for conceptual graphs. In Goichiro Hanaoka and Toshihiro Yamauchi, editors,

Advances in Information and Computer Security, pages 105–122, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[106] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In Ted Wobber and Peter Druschel, editors, *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 85–100. ACM, 2011.

[107] Pycryptodome, 2021. version 3.10.1.

[108] Python Cryptographic Authority. pyca/cryptography, 2023. version 39.0.0.

[109] M. H. Quenouille. Approximate tests of correlation in time-series. *Journal of the Royal Statistical Society. Series B (Methodological)*, 11(1):68–84, 1949.

[110] Open-Data-Plattform öV Schweiz. Fahrplan 2016 (gtfs). `https://opentransportdata.swiss/en/dataset/timetable-2016-gtfs`, November 2016.

[111] Adam Sealfon. Shortest paths and distances with differential privacy. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, page 29–41, New York, NY, USA, 2016. Association for Computing Machinery.

[112] Bin Shao, Haixun Wang, and Yatao Li. Trinity: A distributed graph engine on a memory cloud. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, page 505–516, New York, NY, USA, 2013. Association for Computing Machinery.

[113] Meng Shen, Baoli Ma, Liehuang Zhu, Rashid Mijumbi, Xiaojiang Du, and Jiankun Hu. Cloud-based approximate constrained shortest distance queries over encrypted graphs with privacy protection. *IEEE Transactions on Information Forensics and Security*, 13(4):940–953, 2018.

[114] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 350–364, 2007.

[115] A. Shlosser. On estimation of the size of the dictionary of a long text on the basis of a sample. *Engineering Cybernetics*, (19):97–102, 1981.

[116] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, jun 1983.

[117] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, pages 44–55, 2000.

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

[119] Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. *J. ACM*, 65(4), apr 2018.

[120] Abdel Aziz Taha and Allan Hanbury. An efficient algorithm for calculating the exact hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(11):2153–2163, 2015.

[121] Paul Valiant and Gregory Valiant. Estimating the unseen: Improved estimators for entropy and other properties. In *Advances in Neural Information Processing Systems*, volume 26, pages 2157–2165, 2013.

[122] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 178–195, New York, NY, USA, 2018. Association for Computing Machinery.

[123] Gérard Viennot. Chain and antichain families grids and young tableaux. In Maurice Pouzet and Denis Richard, editors, *Orders: Description and Roles Ordres: Description et Rôles*, volume 99 of *North-Holland Mathematics Studies*, pages 409 – 463. North-Holland, 1984.

[124] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[125] Boyang Wang, Yantian Hou, Ming Li, Haitao Wang, and Hui Li. 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, 2014.

[126] Jiafan Wang and Sherman S. M. Chow. Simple storage-saving structure for volume-hiding encrypted multi-maps. In Ken Barker and Kambiz Ghazinour, editors, *Data and Applications Security and Privacy XXXV*, pages 63–83, Cham, 2021. Springer International Publishing.

[127] Jianfeng Wang, Shi-Feng Sun, Tianci Li, Saiyu Qi, and Xiaofeng Chen. Practical volume-hiding encrypted multi-maps with optimal overhead and beyond. In *Proceedings*

of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS '22, page 2825–2839, New York, NY, USA, 2022. Association for Computing Machinery.

[128] Qian Wang, Kui Ren, Minxin Du, Qi Li, and Aziz Mohaisen. Secgdb: Graph encryption for exact shortest distance queries with efficient updates. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, volume 10322 of *Lecture Notes in Computer Science*, pages 79–97. Springer, 2017.

[129] David J. Wu, Joe Zimmerman, Jérémy Planul, and John C. Mitchell. Privacy-preserving shortest path computation. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[130] Yujie Xue, Lanxiang Chen, Yi Mu, Lingfang Zeng, Fatemeh Rezaeibagha, and Robert H. Deng. Structured encryption for knowledge graphs. *Inf. Sci.*, 605(C):43–70, aug 2022.

[131] Junhua Zhang, Wentao Li, Long Yuan, Lu Qin, Ying Zhang, and Lijun Chang. Shortest-path queries on complex networks: Experiments, analyses, and improvement. *Proc. VLDB Endow.*, 15(11):2640–2652, jul 2022.

[132] Zheguang Zhao, Seny Kamara, Tarik Moataz, and Stan Zdonik. Encrypted databases: From theory to systems. In *Conf. on Innovative Data Systems Research*, CIDR, 2021.