Full Database Reconstruction in Two Dimensions

Francesca Falzon*
University of Chicago
ffalzon@uchicago.edu

Evangelia Anna Markatou*
Brown University
markatou@brown.edu

Akshima
University of Chicago
akshima@uchicago.edu

David Cash
University of Chicago
davidcash@uchicago.edu

Adam Rivkin
University of Chicago
amrivkin@uchicago.edu

Jesse Stern
University of Chicago
jesseastern@uchicago.edu

Roberto Tamassia
Brown University
rt@cs.brown.edu

ABSTRACT

In the past few years, we have seen multiple attacks on one-dimensional databases that support range queries. These attacks achieve full database reconstruction by exploiting access pattern leakage along with known query distribution or search pattern leakage. We are the first to go beyond one dimension, exploring this threat in two dimensions. We unveil an intrinsic limitation of reconstruction attacks by showing that there can be an exponential number of distinct databases that produce equivalent leakage. Next, we present a full database reconstruction attack. Our algorithm runs in polynomial time and returns a poly-size encoding of all databases consistent with the given leakage profile. We implement our algorithm and observe real-world databases that admit a large number of equivalent databases, which aligns with our theoretical results.

CCS CONCEPTS

• **Security and privacy → Cryptanalysis and other attacks**; **Database and storage security**.

KEYWORDS

Encrypted Database; Database Reconstruction; Attack

ACM Reference Format:
Francesca Falzon, Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia. 2020. Full Database Reconstruction in Two Dimensions. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3372297.3417275

1 INTRODUCTION

Encryption can mitigate the risk of a data breach, whether stored in local infrastructure or at a cloud service. However, standard encryption limits the ability of the server holding the ciphertexts to search data unless the decryption key is available. Many solutions have been suggested to enable server processing of encrypted data on behalf of clients. In this work, we consider solutions that allow the server to respond to *range queries* on an encrypted database without decrypting the data. *Order revealing encryption* [1, 4, 5] supports range queries, but various such schemes, including the most-secure "ideal" constructions, have been shown vulnerable to devastating *leakage abuse attacks* (e.g., [3, 13, 20, 31]) that recover data in certain circumstances, a fact that underscores the need to better understand the security of these schemes.

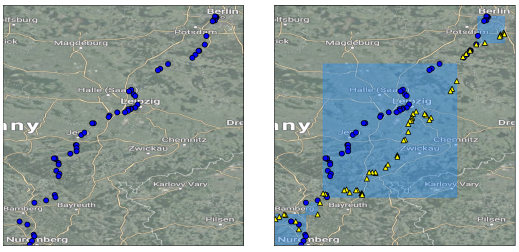


Figure 1: Reconstruction of Malte Spitz's [36] location data from 08/31/2009. Original locations are drawn as blue points and their reflections as yellow triangles. The points in each highlighted box and stand-alone pair can independently flip along the diagonal, producing 1024 equivalent databases.

As an alternative to order revealing encryption, *searchable encryption* involves building an encrypted index to look up range queries (e.g., [10] and the survey by Fuller et al. [15]). These schemes support a variety of query types, including subsets of SQL (e.g. [14, 21]). When a query is processed, the client learns the results of the query and then learns some "leakage" about the data and the query. This approach has been used both in data management research (e.g., [33–35]) and industry (e.g., [9, 32]) and will likely continue to be deployed despite leakage-based attacks because it still resists other weaker attacks (e.g., a smash-and-grab data breach no longer reveals an entire column of credit card information).

Several recent works [18, 19, 23–27] 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '20, November 9–13, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417275

*FF and EAM are co-first authors who contributed equally, listed alphabetically. The remaining authors are listed alphabetically. The Brown and Chicago teams contributed equally to this work.

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.

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 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. [18, 23]) 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 [6, 8, 10], 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., [16]), and dynamic constructions like [7, 11], 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 [37]. 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. [21, 22] 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 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 4, Theorems 4.3 and 4.4).
(2) We **develop** an efficient poly-time full database reconstruction attack that encodes the potentially exponential databases in poly-space (Section 5, Algorithm 3 and Theorem 5.5).

Table 1: Comparison of our work with related FDR attacks.

	Queries			Assumptions		Leakage	
	1D range	1D k-NN	**2D range**	Query distrib.	Data distrib.	Access pattern	Search pattern
Kellaris+ [23]	✓			Uniform	Any	✓	✓
Lacharité+ [26]	✓			Agnostic	Dense	✓	
Kornaropoulos+ [24]		✓		Uniform	Any	✓	
Grubbs+ [18]	✓			Uniform	Any	✓	
Grubbs+ [18]	✓			Uniform	Minor info	✓	
Markatou+ [27]	✓			Agnostic	Any	✓	✓
Kornaropoulos+ [25]	✓	✓		Agnostic	Any	✓	✓
This Work			✓	**Agnostic**	**Any**	✓	✓
This Work			✓	**Known**	**Any**	✓	

(3) We **implement** the attack and **evaluate** it on real-world location and health data (Section 6).
(4) Given access to training data, we show how to **reduce** the size of the solution set (Section 7).

This paper is the product of merging two independent lines of work, [2] and [29].

1.2 Prior and related work

Kellaris et al. [23] show that for a one-dimensional database over domain $[1, N]$, one can determine the exact record values up to reflection with $O(N^4 \log N)$ uniformly random queries. Also, reconstruction can be done 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. In [26], Lacharitè et al. improve on the dense database attack and present an algorithm that succeeds in reconstructing dense databases with $O(N \log N)$ queries. For large N, these query complexities can quickly become impractical, so they additionally presented an *ϵ-approximate database reconstruction* (ϵ-ADR) attack that recovers all plaintext values up to some additive ϵN error with $O(N \log \epsilon^{-1})$ queries.

The *sacrificial ϵ-ADR* approximation attack by Grubbs et al. [18] is scale free, i.e., its success depends only on the value of ϵ (as opposed to both ϵ and N). The first attack issues $O(\epsilon^{-4} \log \epsilon^{-1})$ queries and the second attack succeeds with $O(\epsilon^{-2} \log \epsilon^{-1})$ queries under the assumption that there exists some record in the database whose value is in the range $[0.2N, 0.3N]$ (or its reflection). Both attacks rely on uniform query distribution. The authors also prove that database reconstruction from known queries can be reduced to PAC learning.

Reconstruction attacks from k-NN queries are presented by Kornaropoulos et al. [24]. For cases when exact reconstruction is impossible, they characterize the space of valid reconstructions and give approximation reconstruction methods. In other work, Kornaropoulos et al. [25] combine access pattern leakage with search-pattern leakage and apply statistical learning methods to reconstruct databases with unknown query distributions from range and k-NN queries. Table 1 compares our attacks with selected related work.

There has also been some work on mitigation techniques for leakage-abuse attacks [12, 17, 28, 30].

2 PRELIMINARIES

Domains and databases. For an integer N let $[N] = \{1, 2, \ldots, N\}$. For the rest of the paper, we fix positive integers N_0, N_1 and let $\mathcal{D} = [N_0] \times [N_1]$. 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 $x_0 \le w_0$ and $x_1 \le w_1$. We denote this as $x \preceq w$. Similarly, point $w \in \mathcal{D}$ *anti-dominates* point $x \in \mathcal{D}$ if $w_0 \le x_0$ and $x_1 \le w_1$, and we denote this as $x \preceq_a w$.

We define a *2-dimensional database D over domain $\mathcal{D}$* as an element of $\mathcal{D}^R$ for some integer $R \ge 1$, 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 paper, 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* 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,

$$\mathsf{Resp}(D, q) = \{j \in [R] : c \preceq D[j] \preceq d\}. \tag{1}$$

We define the *response multiset of a database D*, denoted $\mathsf{RM}(D)$, to be the *multiset* of all access patterns of D. Formally,

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

This is a multiset because distinct queries q, q' may have $\mathsf{Resp}(D, q) = \mathsf{Resp}(D, q')$, i.e., return the same records. We also define the corresponding set $\mathsf{RS}(D) = set(\mathsf{RM}(D))$, by removing any duplicate entries from $\mathsf{RM}(D)$.

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.

Leakage Equivalent databases. We say that databases $D, D' \in \mathcal{D}^R$ with the same record identifiers are *equivalent* if $\mathsf{RM}(D) = \mathsf{RM}(D')$, meaning that the response multisets of the databases are

exactly the same. (This implies that they have the same number of records.) We denote the set of databases equivalent to D as

$$\mathsf{E}(D) = \{D' \in \mathcal{D}^R : D \text{ and } D' \text{ are equivalent}\}. \quad (3)$$

Full database reconstruction. We define the problem of *Full Database Reconstruction* (*FDR*) as follows: Given $\mathsf{RM}(D)$ for some database D, compute $\mathsf{E}(D)$.

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.

3 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 2)

$$w'_0 = w_1 \cdot \frac{N_0 + 1}{N_1 + 1}; \qquad w'_1 = w_0 \cdot \frac{N_1 + 1}{N_0 + 1}. \quad (4)$$

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. 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 $\mathcal{D}$.

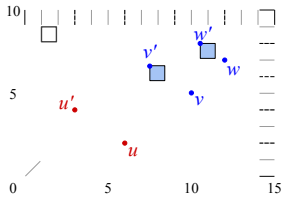


Figure 2: Points u, v and w of domain $\mathcal{D} = [14] \times [9]$ (thick black rectangle) and their reflections u', v' and w'. We have that $u' \in \mathcal{D}$ but $v' \notin \mathcal{D}$ and $w' \notin \mathcal{D}$. By Lemma 3.1, the points of $\mathcal{D}$ whose reflection is in $\mathcal{D}$ have coordinates of the type $(3i, 2j)$, i.e., are at the intersections of the dotted grid-lines.

Lemma 3.1. *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 $(0, N_1 + 1)$.

3.1 Query Densities

We will repeatedly use a strategy that generalizes the main observation of [23]. There, in trying to determine a point x, they observed

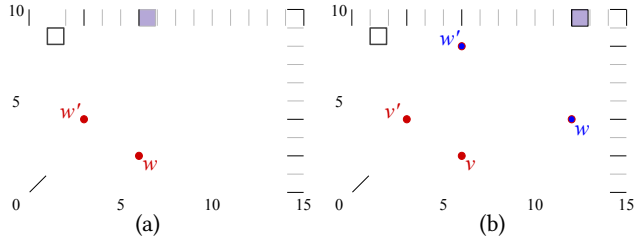


Figure 3: Illustration of Equations 5 and 6 for points of domain $\mathcal{D} = [14] \times [9]$. (a) The query density of point $w = (6, 2)$ is the product of the areas of the two rectangles, i.e., $\rho_w = 6 \cdot 2 \cdot (15 - 6) \cdot (10 - 2) = 12 \cdot 72 = 864$. Since the reflection $w' = (3, 4)$ of w is a point of $\mathcal{D}$, by Lemma 3.2, 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 3.3, we have $\rho_{v,w} = \rho_{v',w} = \rho_{v,w'} = \rho_{v',w'} = 216$.

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.

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

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

Thus, these are the number of queries that contain x or x and y (respectively). 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 and 5).

$$\boxed{\rho_x = x_0 x_1 (N_0 + 1 - x_0)(N_1 + 1 - x_1)} \quad (5)$$

Lemma 3.2. *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.

$$\boxed{\rho_{v,w} = v_0 v_1 (N_0 + 1 - w_0)(N_1 + 1 - w_1)} \quad (6)$$

Again, we obtain the same query density by replacing one or both points of a pair with their reflections, as shown in Lemma 3.3. We note that this equation only holds when $v \preceq w$. If not, there are similar formulas depending on their (anti-) dominance relationship.

Lemma 3.3. *Let $v \preceq w$ be points of domain $\mathcal{D}$ and let v' and w' be their reflections. We have*

$$\rho_{v,w} = \rho_{v',w} \quad \text{if } v' \in \mathcal{D} \text{ and } v' \preceq w$$
$$\rho_{v,w} = \rho_{v,w'} \quad \text{if } w' \in \mathcal{D} \text{ and } v \preceq w'$$
$$\rho_{v,w} = \rho_{v',w'} \quad \text{if } v' \in \mathcal{D},\ w' \in \mathcal{D} \text{ and } v' \preceq w'$$

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_0N_1. The second lemma identifies when points can be solved for, possibly up to a reflection symmetry.

Lemma 3.4. *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 $\rho_x = x_0x_1(N_0 + 1 - x_0)(N_1 + 1 - x_1)$. For each $\beta \in [N_0]$, when we set $x_0 = \beta$ we get: $\alpha = \beta x_1(N_1 + 1 - x_1)(N_0 + 1 - \beta) \implies$

$$\boxed{x_1^2 - (N_1 + 1)x_1 + \frac{\alpha}{\beta(N_0 + 1 - \beta)} = 0} \tag{7}$$

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 $\alpha = x_0\gamma(N_1 + 1 - \gamma)(N_0 + 1 - x_0) \implies$

$$\boxed{x_0^2 - (N_0 + 1)x_0 + \frac{\alpha}{\gamma(N_1 + 1 - \gamma)} = 0} \tag{8}$$

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 3.5 in Figure 4.

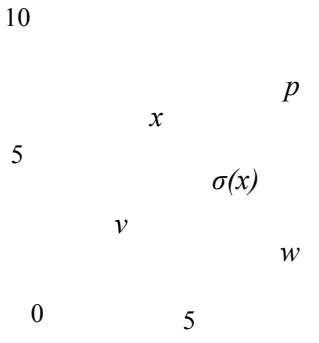


Figure 4: In Lemma 3.5, we know points v and w, and want to determine point x. The grey (solid), green (dashed-dotted) and purple (dashed) lines denote curves ρ_x, $\rho_{v,x}$ and $\rho_{w,x}$, respectively. The intersection of these three curves returns a unique location for x. To demonstrate that 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 3.5. *Let $v, w \in [N_0] \times [N_1]$ and let $\alpha, \beta, \gamma \in \mathbb{Z}$. Then the system of equations*

$$\begin{aligned} \rho_x &= \alpha \\ \rho_{v,x} &= \beta \\ v &\preceq x \end{aligned} \tag{9}$$

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

$$\begin{aligned} \rho_{w,x} &= \gamma \\ w &\preceq_a x \end{aligned} \tag{10}$$

then the system has at most one integral solution. Similarly, the system has at most one solution if the last equation of Systems (9) and (10) 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 9 as

$$\begin{aligned} \alpha = \rho_x &= x_0x_1(N_0 + 1 - x_0)(N_1 + 1 - x_1) \\ \beta = \rho_{v,x} &= v_0v_1(N_0 + 1 - x_0)(N_1 + 1 - x_1) \end{aligned}$$

We then rewrite the above equations as

$$\boxed{\begin{aligned} \frac{\beta}{v_0v_1} &= (N_0 + 1 - x_0)(N_1 + 1 - x_1) \\ \mu = \alpha\frac{v_0v_1}{\beta} &= x_0x_1 \\ \omega = -\frac{\beta}{v_0v_1} + N_0N_1 + N_0 + N_1 + 1 + \mu &= (N_1 + 1)x_0 + (N_0 + 1)x_1. \end{aligned}} \tag{11}$$

Using substitution and the quadratic formula we can obtain the following two solutions for System 11:

$$\begin{aligned} \hat{x}' &= \left(\frac{\omega - \mathrm{sqrt}}{2(N_1 + 1)}, \frac{\omega + \mathrm{sqrt}}{2(N_0 + 1)}\right) \\ \hat{x}'' &= \left(\frac{\omega + \mathrm{sqrt}}{2(N_1 + 1)}, \frac{\omega - \mathrm{sqrt}}{2(N_0 + 1)}\right) \end{aligned} \tag{12}$$

where $\mathrm{sqrt} = \sqrt{\omega^2 - 4\mu(N_1 + 1)(N_0 + 1)}$. Note that $\hat{x}'$ and $\hat{x}''$ are reflections across the main diagonal. Moreover, note that by Lemmas 3.2 and 3.3 we know that $\rho_x = \rho_{\sigma(x)}$ and $\rho_{v,x} = \rho_{v,\sigma(x)}$. Thus, both x' and x'' solve System (9) 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 (9) would not be satisfied and the lemma follows.

Let us now consider the additional equations in System (10). Applying the formula for ρ given that $w \preceq_a x$ yields

$$\gamma = \rho_{w,x} = x_0w_1(N_0 + 1 - w_0)(N_1 + 1 - x_1).$$

We can then rearrange to obtain the system of equations

$$\boxed{\begin{aligned} \frac{\gamma}{(N_0 + 1 - w_0)w_1} &= N_1x_0 + x_0 - x_0x_1 \\ -\frac{\beta}{v_0v_1} + N_0N_1 + N_0 + N_1 + 1 &= N_1x_0 - N_0x_1 + x_0x_1 \end{aligned}} \tag{13}$$

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 (9) and (10) are replaced with $x \preceq_a v$ and $x \preceq w$, respectively. By applying the rho equations we see that

$$\begin{aligned} \beta = \rho_{v,x} &= v_0x_1(N_0 + 1 - x_0)(N_0 + 1 - v_0) \\ \gamma = \rho_{w,x} &= x_0x_1(N_0 + 1 - w_0)(N_1 + 1 - w_1) \end{aligned}$$

which we can rearrange to get the system of equations

$$\begin{aligned} \frac{\gamma}{(N_0+1-w_0)(N_1+1-w_1)} &= x_0x_1 \\ \frac{\beta}{v_0(N_1+1-v_1)} &= N_0x_1+x_1-x_0x_1. \end{aligned} \tag{14}$$

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

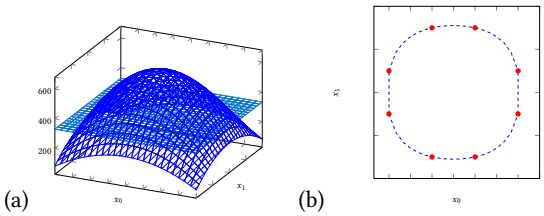
We make the standard assumption that arithmetic on numbers of size (number of bits) $O(\log N_0N_1)$ can be done in constant time. We also employ full-precision arithmetic and use symbolic representations for non-integer values (e.g., square root of a number that is not square).

3.2 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 [23] 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 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.

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 paper. 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



(a) (b)

Figure 5: Solving $\rho_x = \alpha$ (Equation 5): (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.

in Figure 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 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 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.

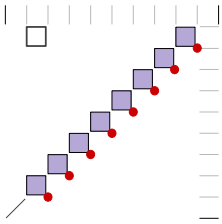


Figure 6: Reflecting any subset of the 8 database points yields an equivalent database. Further applying rigid motions, we get a total of $8 \times 2^8 = 1,024$ equivalent databases.

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; 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 3.4 and 3.5, 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 $\mathsf{E}(D)$.

4 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 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 begin understanding how these equivalent databases are formed, define a *component* of D to be 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 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 4.1. *Any database can be uniquely partitioned into components.*

The proof to Lemma 4.1 can be found in the Appendix.

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 7, states that replacing the points of a reflectable component with their reflections leaves the search pattern unchanged.

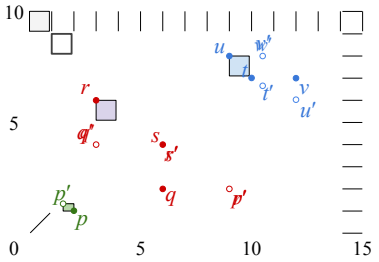


Figure 7: A database D over domain $[14] \times [9]$. Solid circles represent database points and hollow circles represent their reflections. D has components $C_1 = \{p\}$, $C_2 = \{q, r, s\}$, and $C_3 = \{t, u, v\}$. C_1 and C_3 are nonreflectable while C_2 is reflectable. Replacing the points of C_2 with their reflections yields a database equivalent to D.

Lemma 4.2. *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.*

The proof of Lemma 4.2 can be found in the Appendix and its main argument is schematically illustrated in Figure 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 4.2 to build a database that admits an exponential number of equivalent databases (see Figure 6).

Theorem 4.3. *For every integer R, there exists a family of 2^R databases with R points that are equivalent to each other.*

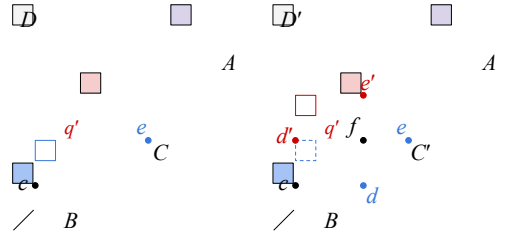


Figure 8: Case 3 of the proof of Lemma 4.2: database D' is obtained from D by reflecting component C to yield C'; the blue range query (c, e) on D and the red range query (c, e') on D' return the same response, as seen from the intersection (c, f) and differences, (d, e) and (d', e'), of the two ranges.

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 Lemma 4.2, 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 4.4. *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 of Theorem 4.4 can be found in the Appendix. This 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.

5 FULL DATABASE RECONSTRUCTION

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

5.1 Overview of the attack

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

(1) We identify the extreme points of the database, including any corner points. (Algorithm 1)
(2) We 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 2)
(3) We 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 4)
(4) Using the recovered locations, we partition the database into components. (Algorithm 5)
(5) We prune the locations for the points in each partition down to one location per point. (Algorithm 3)

5.2 Preprocessing

Before we delve into the algorithm, we will preprocess the input. Given multiset $\mathrm{RM}(D)$, we generate the corresponding set $\mathrm{RS}(D)$. At this point, we would like to note that $\mathrm{RM}(D)$ has size $O((N_0 N_1)^2) = O(N^2)$ and $\mathrm{RS}(D)$ 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 $\mathrm{RS}(D)$ and time $O(RN^2)$ to read $\mathrm{RM}(D)$.

We also preprocess $\mathrm{RM}(D)$ and $\mathrm{RS}(D)$ making sure that each value in the domain corresponds to at most one identifier. We do

that by finding the smallest set S in $\mathrm{RS}(D)$ that contains a given ID. Then, we go through $\mathrm{RM}(D)$ and $\mathrm{RS}(D)$ replacing set S from each response with a new identifier.

5.3 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 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 1 takes as input $\mathrm{RS}(D)$ 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 $\mathrm{RS}(D)$, 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 $\mathrm{RS}(D)$ 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 $\mathrm{RS}(D)$ that contains all points in this subset is the largest one. The algorithm then returns all possible configurations of this subset of points.

Lemma 5.1. *Let D be a database with R records and let $\mathrm{RS}(D)$ be its response set. Algorithm 1 returns all possible configurations of extreme points of D (up to symmetries) in time $O(\min(R^5, RN^2))$.*

The proof of Lemma 5.1 can be found in the Appendix.

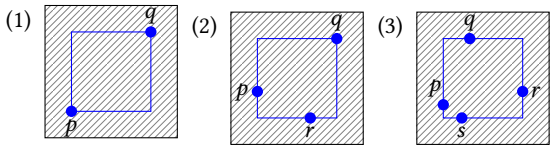


Figure 9: Three mutually exclusive cases for a minimal set of extreme points of a database: (1) two corners, p and q; (2) one corner, q; (3) no corners.

Algorithm 1: *FindExtremes*($\mathrm{RS}(D)$)

Let L be the largest set in $\mathrm{RS}(D)$ and Edges, PosExtremes empty lists
Let S_1 be the 2^{nd} largest set in $\mathrm{RS}(D)$. Add $L - S_1$ to Edges.
Let S_2 be the 2^{nd} largest set that contains $L - S_1$. Add $L - S_2$ to Edges.
Let S_3 be the 2^{nd} largest set that contains $L - S_1$ and $L - S_2$ (if it exists). Add $L - S_3$ to Edges.
Let S_4 be the 2^{nd} largest set that contains $L - S_1$, $L - S_2$ and $L - S_3$ (if it exists). Add $L - S_4$ to Edges.
Let PosExtremes be an empty list.
for all $E \in$ Edges **do**
  if $|E| = 1$ **then**
    Add the one ID in E to PosExtremes
  else
    Consider the responses of size 2 in $\mathrm{RS}(D)$ containing only IDs of E. Count in how many such responses each ID of E appears and add any IDs that appear exactly once to PosExtremes
for $i = 2$ **to** 4 **do**
  for all subsets of IDs $E \subseteq$ PosExtremes such that $|E| = i$ **do**
    if L is the only set in $\mathrm{RS}(D)$ containing E **then**
      { The i IDs of E refer to a minimal set of extreme points }
      Let *PosConfigs* be an empty list
      for all possible configurations of extreme point IDs in E **do**
        Add a hashmap to *PosConfigs* with entries mapping keys left, right, bot, and top, to their respective IDs of E.
      return *PosConfigs*

5.4 Segment the database

Given a set of extreme points *left*, *right*, *bot*, and *top* computed by Algorithm 1, 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 5.3), 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 10.

Algorithm 2 (*Segmentation*) takes as input $\mathrm{RS}(D)$ and *PosConfigs* (output by Algorithm 1) and returns a list of tuples, where each tuple comprises of two IDs, for *left* and *right*, and three sets of IDs.

Algorithm 2: *Segmentation*(*PosConfigs*, $\mathrm{RS}(D)$)

Let *Segmentations* be an empty list.
for all hashmaps $H \in$ *PosConfigs* **do**
  Let S_2 be the smallest set in $\mathrm{RS}(D)$ containing $H[\text{left}]$ and $H[\text{right}]$
  Let T be the smallest set in $\mathrm{RS}(D)$ containing $H[\text{top}]$, $H[\text{left}]$ and $H[\text{right}]$
  $S_1 = T - S_2$
  $S_3 = D - T$
  Add $((H[\text{left}], H[\text{right}]), (S_1, S_2, S_3))$ to *Segmentations*.
return *Segmentations*

Lemma 5.2. *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 1 on $\mathsf{RS}(D)$. Then Algorithm 2 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.*

The proof of Lemma 5.2 can be found in Appendix B.5.

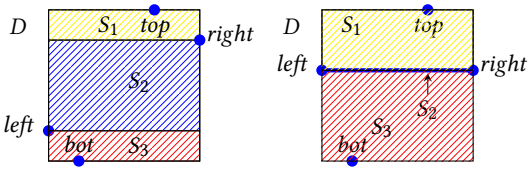


Figure 10: Segmenting the database into three sets of points.

5.5 Find candidate locations

Algorithm 4 finds 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 5.3. 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 3.4. We then use a second parameter, $\rho_{left,right}$, to obtain the set of equations in System 9, 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 3.5. The pseudocode for computing the possible values of each database can be found in Algorithm 4 in Apendix B.

Lemma 5.3. *Let $D \in \mathcal{D}^R$ be a database with R records and let $\mathsf{RM}(D)$ be its response multiset. Algorithm 4 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 $\mathrm{ID} \in [R]$, we have $\hat{D}[\mathrm{ID}] \in H[\mathrm{ID}]$.*

The proof of Lemma 5.3 can be found in Appendix B.6.

5.6 Partition a database into components

Algorithm 5 (*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 3.

For a record ID of database D, let $low(\mathrm{ID})$ and $high(\mathrm{ID})$ be the lower and higher orthogonal projections of point $v = D[\mathrm{ID}]$ on the main diagonal, respectively (see Figure 11), i.e., $low(\mathrm{ID})$ and $high(\mathrm{ID})$ are the intersections of the main diagonal with horizontal and vertical lines through point v, where $low(\mathrm{ID}) \preceq high(\mathrm{ID})$. Clearly, a point and its reflection have the same orthogonal projections on the main diagonal.

Algorithm 5 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 11). The algorithm finds the reflectable components by "walking up"

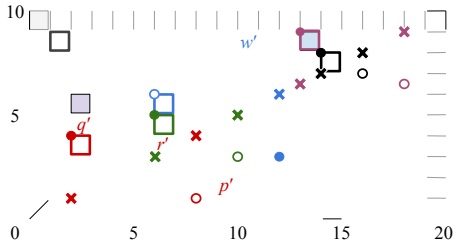


Figure 11: Partitioning a database with 5 points (filled circles) over domain $[19]\times[9]$ into a reflectable component (bottom left) and a nonreflectable component (top right) using Algorithm 5. Reflections of points are depicted as empty circles, and projections on the main diagonal as cross marks.

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., $p = D[\mathrm{ID}]$), and finally IDs such that p is the lower projection. The pseudocode for this algorithm, Algorithm 5 can be found in the Appendix.

Lemma 5.4. *Given a database D with R identifiers, each with one or two possible locations, $p, \sigma(p)$, from domain $[N_0] \times [N_1]$, Algorithm 5 (Partition) partitions D into its reflectable and nonreflectable components in time $O(\min(R \log R, R + N_0, R + N_1))$.*

The proof of Lemma 5.4 can be found in the Appendix.

5.7 Prune the candidate reconstructions

Algorithm 3 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 Algorithm 5, 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 $(low(\mathrm{ID}_1), high(\mathrm{ID}_1))$ and $(low(\mathrm{ID}_2), high(\mathrm{ID}_2))$ intersect in more that one points. More formally, when $high(\mathrm{ID}_1) > low(\mathrm{ID}_2)$ and $high(\mathrm{ID}_2) > low(\mathrm{ID}_1)$. Any identifiers for which $high(\mathrm{ID}) = low(\mathrm{ID})$ are ignored for the purposes of G. The construction of graph G is illustrated in Figure 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 $(\mathrm{ID}_1, \mathrm{ID}_2)$, 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 $(\mathrm{ID}_1, \mathrm{ID}_2)$. In case none are consistent, then this database family is invalid.

Algorithm 3 takes as input D's response multiset, $\mathsf{RM}(D)$, and returns a list of families of databases. Each family consists of a

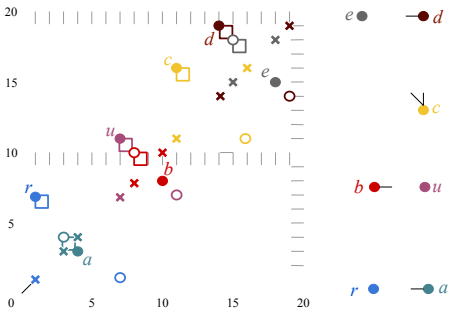


Figure 12: (left) Database with 7 records and 3 components over domain $[19]\times[19]$. The true points are shown with filled circles, their reflections with empty circles, and their projections on the main diagonal with cross marks. (right) Graphs for the components constructed by Algorithm 3, where an edge between two records indicates that fixing the point of one record fixes the point of the other record.

database $\hat{D}$ and its partition into components. The attacker produces $E(D)$ using the output of Algorithm 3. Theorem 5.5 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 5.5. *Given the response multiset* $\mathsf{RM}(D)$ *of some database* D *with* R *records over domain* $[N_0]\times[N_1]$, *Algorithm 3* (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* $N=N_0N_1$.

The proof of Theorem 5.5 can be found in the Appendix. We recall from Section 5.2 that reading the input to the algorithm takes time $O(RN^2)$.

6 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)*, seven years, 2012-2018, of Chicago crime locations from the City of Chicago's data portal,† and the mobile phone records of Malte Spitz, a German Green party politician.‡

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 [19, 23]. 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. More details on how we chose our attributes can be found in Appendix C.

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 $w_0=\frac{(lat-lat_{min})\cdot(N_0-1)}{lat_{max}-lat_{min}}+1$

Algorithm 3: FDR(RM(D))

```
1:  Databases = [ ], RS(D) is the set of RM(D)
2:  PosConfigs = FindExtremes(RS(D))(Algorithm 1)
3:  Segmentations = Segmentation(PosConfigs, RS(D)) (Algorithm 2)
4:  Solutions = Solve(PosConfigs, Segmentations, RM(D)) (Algorithm 4)
5:  for D̂ in Solutions do
6:    Partition = Partition(D̂) (Algorithm 5)
7:    for each (Component, refl) in Partition do
8:      Let Resolved be the set of ID ∈ Component s.t. |D̂[ID]| = 1
9:      Let Unresolved be the set of ID ∈ Component s.t. |D̂[ID]| = 2
10:     if Unresolved ≠ ∅ AND (Resolved = ∅ OR Resolved only contains
        points on the diagonal) then
11:       Pick random ID ∈ Unresolved and remove one entry of D̂[ID]
12:       Remove ID from Unresolved and add it to Resolved
13:
14:     Construct graph G with nodes all ID ∈ Component, ignoring
        points on the diagonal. There is an edge between ID1, ID2 in G, if
        the boxes defined by (low(ID1), high(ID1)) and (low(ID2),
        high(ID2)) 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 ID1 ∈ Resolved.
17:     for each (ID1, ID2) ∈ Edges(G) do
18:       // At each iteration, one ID is added to Resolved or the current
          solution D̂ is discarded
19:       if ID2 ∈ Unresolved then
20:         Let D̂[ID1] = r
21:         m = the number of responses in RM(D) containing ID1 & ID2
22:         if r ∈ D̂[ID2] then
23:           Remove r from D̂[ID2]
24:         for u in D̂[ID2] do
25:           Compute ρu,r using Equation 6
26:           if ρu,r ≠ m then
27:             Remove u from D̂[ID2]
28:
29:         if |D̂[ID2]| = 1 then
30:           Add ID2 to Resolved and remove it from Unresolved
31:         else
32:           // Here, D̂[ID2] = ∅
33:           Go to line 5 (discard the current solution D̂)
34:   Compute the response multiset RM(D̂)
35:   if RM(D̂) = RM(D) then
36:     Add D̂, Partition to Databases
37: Return Databases
```

*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.

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

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

Table 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, $i = 0, 1, 2, \ldots$.

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

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 $c \leq d$.

Our findings. Our results are shown in Table 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 one fourth of the instances had ≥ 2 reflectable components, with a maximum of eight components. To illustrate the structure of the Spitz data, we show in Figure 1 the phone record locations for 08/31/2009 and our reconstruction of them[§]. 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 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.

7 AUTOMATICALLY FINDING D IN $\mathsf{E}(D)$

Our attack in Section 5, and those of prior work [19, 23] 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

[§]Figure 1 map source: Google Maps

Section 5 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 $\mathcal{D}_i$ 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.

We run experiments on both 1D and 2D databases. We present our results in one dimension on just HCUP data in Table 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.

Table 3: Symmetry breaking for 1D range queries.

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

We present our two-dimensional results in Table 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.

Table 4: Symmetry breaking for 2D range queries.

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

8 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.

A first direction of followup research is to tighten the bound on the number of seed databases of the reconstruction. Theorem 4.4 gives a linear bound (in the perimeter of the domain) and we suspect one could instead prove a logarithmic bound. Another direction is to relax the assumptions on the amount of leakage available to the adversary. Our attack requires complete information and produces an exact reconstruction. It would be interesting to develop approximate reconstruction attacks from partial information (e.g., a subset of all the query responses or just their sizes). Finally, devising attacks for databases of arbitrary dimension is an important open problem. It is easy to see that our exponential lower bound on the size of a family of equivalent databases (Theorem 4.3) extends to higher dimensions. However, additional techniques may be needed to extend our reconstruction approach to higher dimensions.

9 ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable comments and Evgenios Kornaropoulos for providing insights in early stages of this work. This work was supported in part by NSF grants CNS 1928767 and CNS 1925288, and by the Kanellakis Fellowship at Brown University.

REFERENCES

[1] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order Preserving Encryption for Numeric Data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 2004)*.

[2] Akshima, David Cash, Francesca Falzon, Adam Rivkin, and Jesse Stern. 2020. Multidimensional Database Reconstruction from Range Query Access Patterns. Cryptology ePrint Archive, Report 2020/296. (2020). https://eprint.iacr.org/2020/296.

[3] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. 2018. The Tao of Inference in Privacy-Protected Databases. *Proc. VLDB Endow.* 11, 11 (July 2018), 1715–1728.

[4] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. 2009. Order-Preserving Symmetric Encryption. In *Advances in Cryptology - EUROCRYPT 2009*.

[5] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. 2011. Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. In *Advances in Cryptology – CRYPTO 2011*.

[6] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic searchable encryption in very-large databases: data structures and implementation. In *21st Annual Network and Distributed System Security Symposium 2014 (NDSS 2014)*.

[7] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. In *Proc. of ACM Conf. on Computer and Communications Security 2018 (CCS 2018)*.

[8] Melissa Chase and Seny Kamara. 2010. Structured Encryption and Controlled Disclosure. In *Advances in Cryptology – ASIACRYPT 2010*.

[9] Ciphercloud. 2020. CipherCloud: Cloud Data Security Company. (2020). http://www.ciphercloud.com Accessed on May 3, 2020.

[10] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2011. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. *Journal of Computer Security* 19, 5 (2011), 895–934.

[11] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2020. Dynamic Searchable Encryption with Small Client Storage. In *27th Annual Network and Distributed System Security Symposium 2020 (NDSS 2020)*.

[12] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. 2020. SEAL: Attack Mitigation for Encrypted Databases via Adjustable Leakage. In *29th USENIX Security Symposium (USENIX Security 20)*.

[13] F. Betül Durak, Thomas M. DuBuisson, and David Cash. 2016. What Else is Revealed by Order-Revealing Encryption?. In *Proc. ACM Conf. on Computer and Communications Security 2016 (CCS 2016)*.

[14] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. 2015. Rich Queries on Encrypted Data: Beyond Exact Matches. In *20th European Symposium on Research in Computer Security 2015 (ESORICS 2015)*.

[15] B. Fuller, M. Varia, A. Yerukhimovich, E. Shen, A. Hamlin, V. Gadepally, R. Shay, J. D. Mitchell, and R. K. Cunningham. 2017. SoK: Cryptographically Protected Database Search. In *Proc. IEEE Symposium on Security and Privacy 2017 (S&P 2017)*.

[16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. 2016. TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption. In *Advances in Cryptology - CRYPTO 2016*.

[17] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. 2020. Pancake: Frequency Smoothing for Encrypted Data Stores. In *29th USENIX Security Symposium (USENIX Security 20)*.

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

[19] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2018. Pump Up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In *Proc. ACM Conf. on Computer and Communications Security 2018 (CCS 2018)*.

[20] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart. 2017. Leakage-Abuse Attacks against Order-Revealing Encryption. In *Proc. IEEE Symp. on Security and Privacy 2017 (S&P 2017)*.

[21] Seny Kamara and Tarik Moataz. 2018. SQL on Structurally-Encrypted Databases. In *Advances in Cryptology – ASIACRYPT 2018*.

[22] Seny Kamara, Tarik Moataz, Stan Zdonik, and Zheguang Zhao. 2020. An Optimal Relational Database Encryption Scheme. Cryptology ePrint Archive, Report 2020/274. (2020). https://eprint.iacr.org/2020/274.

[23] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *Proc. ACM Conf. on Computer and Communications Security 2016 (CCS 2016)*.

[24] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2019. Data Recovery on Encrypted Databases With k-Nearest Neighbor Query Leakage. In *Proc. IEEE Symp. on Security and Privacy 2019 (S&P 2019)*.

[25] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2020. 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)*.

[26] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2018. Improved reconstruction attacks on encrypted data using range query leakage. In *Proc. IEEE Symp. on Security and Privacy 2018 (S&P 2018)*.

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

[28] Evangelia Anna Markatou and Roberto Tamassia. 2019. Mitigation Techniques for Attacks on 1-Dimensional Databases that Support Range Queries. In *Proc. Int. Conf on Information Security 2019 (ISC 2019)*.

[29] Evangelia Anna Markatou and Roberto Tamassia. 2020. Database Reconstruction Attacks in Two Dimensions. Cryptology ePrint Archive, Report 2020/284. (2020). https://eprint.iacr.org/2020/284.

[30] Charalampos Mavroforakis, Nathan Chenette, Adam O'Neill, George Kollios, and Ran Canetti. 2015. Modular Order-Preserving Encryption, Revisited. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD 2015)*.

[31] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *Proc. ACM Conf. on Computer and Communications Security 2015 (CCS 2015)*.

[32] Skyhigh Networks. 2020. Skyhigh Networks. (2020). https://www.skyhighnetworks.com accessed on May 3, 2020.

[33] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. 2016. Big Data Analytics over Encrypted Datasets with Seabed. In *12th USENIX Symposium on Operating Systems Design and Implementation 2016 (OSDI 2016)*.

[34] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2019. Arx: An Encrypted Database using Semantically Secure Encryption. *Proc. VLDB Endow.* 12, 11 (August 2019), 1664–1678.

[35] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proc. of the Twenty-Third ACM Symposium on Operating Systems Principles 2011 (SOSP '11)*.

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

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

A PSEUDOCODE

A.1 Pseudocode of Algorithm 4

Algorithm 4: *Solve*(*Segmentations*, RM(D))

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

// First, find all possible solutions for extreme points.
for all $((\mathrm{ID}_1, \mathrm{ID}_2), (S_1, S_2, S_3)) \in Segmentations$ **do**
  Initialize list L.
  for $left_0 = 1, \ldots, N_0$ **do**
    Compute solutions $left'_1, left''_1$ of Eq. 7, $\beta = left_0$, $\alpha = \mathrm{Rho}[\mathrm{ID}_1]$.
    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. 8, $\gamma = left_1$, $\alpha = \mathrm{Rho}[\mathrm{ID}_2]$.
    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 (11) with $\alpha = \mathrm{Rho}[\mathrm{ID}_2]$, $\beta = \mathrm{Rho}[(\mathrm{ID}_1, \mathrm{ID}_2)]$, and $v = left$.
    if $right' \in \mathcal{D}$ **then** add $((\mathrm{ID}_1, \mathrm{ID}_2), (left, right'), (S_1, S_2, S_3))$ to *Extremes*.
    if $right'' \in \mathcal{D}$ **then** add $((\mathrm{ID}_1, \mathrm{ID}_2), (left, right''), (S_1, S_2, S_3))$ to *Extremes*.

// Now find at most two solutions for all other points.
for all $((\mathrm{ID}_1, \mathrm{ID}_2), (left, right), (S_1, S_2, S_3)) \in Extremes$ **do**
  Initialize hashmap H.
  for all ID $\in S_1$ **do**
    Compute solution *ans* to System (13) with $\beta = \mathrm{Rho}[(\mathrm{ID}_1, \mathrm{ID})]$, $\gamma = \mathrm{Rho}[(\mathrm{ID}_2, \mathrm{ID})]$, $v = left$, $w = right$.
    if $ans \notin \mathcal{D}$ **then go to** line 28.
    else Let $H[\mathrm{ID}_2] = \{ans\}$.
  for all ID $\in S_2$ **do**
    Compute solutions ans, ans' using System (11) with $\alpha = \mathrm{Rho}[\mathrm{ID}]$, $\beta = \mathrm{Rho}[(\mathrm{ID}_1, \mathrm{ID})]$, and $v = left$.
    if $ans \in \mathcal{D}$ **then** Let $H[\mathrm{ID}] = \{ans\}$.
    if $ans' \in \mathcal{D}$ **then** Let $H[\mathrm{ID}] = H[\mathrm{ID}] \cup \{ans'\}$.
    if $(ans, ans') \notin \mathcal{D}^2$ **then go to** line 28.
  for all ID $\in S_3$ **do**
    Compute solution *ans* to System (14) with $\beta = \mathrm{Rho}[(\mathrm{ID}_1, \mathrm{ID})]$, $\gamma = \mathrm{Rho}[(\mathrm{ID}_2, \mathrm{ID})]$, $v = left$, $w = right$.
    if $ans \notin \mathcal{D}$ **then go to** line 28.
    Let $H[\mathrm{ID}] = \{ans\}$.
  Add H to *Solutions*.
return *Solutions*.

A.2 Pseudocode of Algorithm 5

Algorithm 5: *Partition*(D)

Partition = [ ] { list of components of database D }
Projections = [ ] { list of projections of points on main diagonal }
Let M be an empty hashmap { maps projections to IDs }
for all ID $\in D$ **do**
  $p = D[\mathrm{ID}]$
  Add $low(p)$ and $high(p)$ to *Projections*
  Add ID to $M[low(p)]$; Add ID to $M[high(p)]$
Component = $\emptyset$ { set of IDs of points of current component }
SeenOnce = $\emptyset$ { set of IDs of points of current component for which only one projection has been seen }
Sort list *Projections* by ascending order.
for all $p \in Projections$ **do**
  Order the items ID $\in M[p]$ as follows:
    first, all ID such that $high(\mathrm{ID}) = p \neq low(\mathrm{ID})$,;
    next, all ID such that $high(\mathrm{ID}) = p = low(\mathrm{ID})$;
    finally, all ID such that $high(\mathrm{ID}) \neq p = low(\mathrm{ID})$;
    within each group, order by ID value.
  for all ID $\in M[p]$ **do**
    if ID $\in$ *SeenOnce* **then**
      Remove ID from *SeenOnce*
      if *SeenOnce* $= \emptyset$ **then**
        if *Component* contains a nonreflectable point or a single diagonal point **then**
          refl = *false*
        else
          refl = *true*
        Add (*Component*, *refl*) to *Partition*
        Set *Component* $= \emptyset$
      else
        Add ID to *SeenOnce* and to *Component*
return *Partition*

B PROOFS

B.1 Proof of Lemma 4.1

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. □

B.2 Proof of Lemma 4.2

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 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 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 $\mathsf{RM}(D) = \mathsf{RM}(D')$ and thus D and D' are equivalent. □

B.3 Proof of Theorem 4.4

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 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 3.4, 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 3.5, 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 *left*, *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 *left*, *right* as D', can be obtained from D' by diagonally reflecting components of D'. By Lemma 4.2, 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 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 3.5 (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 3.5, 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 3.5 (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. □

B.4 Proof of Lemma 5.1

Proof. First we argue that after line 5 the set Edges must contain the extreme points. The second largest query in $\mathsf{RM}(D)$ 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 $\mathsf{RM}(D)$ 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 $left, right, bot, top \in D$ 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 $left_0 \le p_0 \le right_0$ and $bot_1 \le p_1 \le top_1$. Let $q = ((left_0, bot_1), (right_0, top_1))$. Then, by definition, the query q returns all $p \in D$ such that

$$(left_0, bot_1) \preceq p \preceq (right_0, top_1)$$

i.e. all of D. For the backward direction, suppose that the minimal query containing $left, right, bot, top \in D$ 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

$$p_0 < \min(left_0, right_0, bot_0, top_0).$$

But that means that $(left_0, bot_1) \not\preceq p$ and hence p cannot be in the minimal query containing those four points, which is a contradiction. Hence an element of $\{left_0, right_0, bot_0, top_0\}$ 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 PosExtremes only contains points that are singular in an edge or potential corners, then $|\mathsf{PosExtremes}| \le 8$, which implies that the for loops on lines 12 and 13 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 1 runs in time $O(R|\mathsf{RS}(D)|) = O(\min(R^5, RN^2))$. □

B.5 Proof of Lemma 5.2

Proof. Note that for all $p \in D$, $left_0 \le p_0 \le right_0$ and $bot_1 \le p_1 \le top_1$. 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 $\mathsf{RM}(D)$ 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 $O(R|\mathsf{RS}(D)| + R) = O(\min(R^5, RN^2))$. □

B.6 Proof of Lemma 5.3

Proof. By Lemma 5.1 we know that Algorithm 1 correctly outputs the configurations of the extreme points, *PosConfigs*. The for loop on line 3 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 12 to 25, Algorithm 4 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 3.4. By Lemma 3.5 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 (11), (13), or (14), respectively. Since Algorithm 4 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 $\mathrm{ID} \in [R]$, we have that $\hat{D}[\mathrm{ID}] \in H[\mathrm{ID}]$.

Computing the necessary query density functions takes time $O(RN^2)$. The loop on line 12 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 28 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 $O(RN^2)$. □

B.7 Proof of Lemma 5.4

Proof. By Lemma 4.1 we know that database D has a unique partition. We shall show that Algorithm 5 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 $high(q)$, $low(q) \preceq high(p), low(p)$, 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 5 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 5 returns a valid partition of the database identifiers. Regarding the running 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 then generated traversing up the diagonal and doing $O(1)$ work per projection. Thus, Algorithm 5 takes $O(\min(R \log R, R + N_0, R + N_1))$ time.

□

B.8 Proof of Theorem 5.5

Proof. Line 35 of Algorithm 3 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 4.4 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 5.1 we can identify the extreme points and all their possible configurations. Lemmas 3.4 and 3.5 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.
(2) Given the solutions for two extreme points, by Lemma 5.3 and Lemma 3.5 we can identify one or two solutions per identifier. Then, by Lemma 5.4 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 $\mathsf{RM}(D)$ that contain both u and r. We have two cases:

(a) The boxes intersect : As an example from Figure 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 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 3 needs to preprocess the leakage, which takes $O(RN^2)$ time. Algorithm 3 runs Algorithms 1, 2 and 4. This takes $O(RN^2 + \min(R^5, RN^2)) = O(RN^2)$. Then, for each database family, we run Algorithm 5 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 $\mathsf{RM}(D)$. 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 5.1, 5.2, 5.3, and 5.4, we have that Algorithm 3 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 $O((N_0+N_1)(RN^2 + R \log R))$. □

C SUPPLEMENTARY CONTENT FOR EXPERIMENTS

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 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 5.

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 ≥ 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 5: HCUP attributes

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 of the districts. For a chosen latitude domain N_0, the minimum longitude domain for a district was typically around $\frac{N_0}{5}$ and the maximum around $3.6N_0$.