Full Database Reconstruction with Access and Search Pattern Leakage

Evangelia Anna Markatou and Roberto Tamassia

Brown University, Providence RI 02912, USA,
markatou@brown.edu, rt@cs.brown.edu

Abstract. The widespread use of cloud computing has enabled several database providers to store their data on servers in the cloud and answer queries from those servers. In order to protect the confidentiality of the data stored in the cloud, a database can be stored in an encrypted form and all queries can be executed on top of the encrypted database. Recent research results suggest that a curious cloud provider may be able to decrypt some of the items in the database after seeing a large number of queries and their (encrypted) results.

In this paper, we focus on one-dimensional databases that support range queries and develop an attack that can achieve full database reconstruction, inferring the exact value of every element in the database. Previous work on full database reconstruction depends on a client issuing queries uniformly at random.

Let N be the number of elements in the database. Our attack succeeds after the attacker has seen each of the possible query results at least once, independent of their distribution. For the sake of query complexity analysis and comparison with relevant work, if we assume that the client issues queries uniformly at random, we can decrypt *the entire* database after observing $O(N^2 \log N)$ queries with high probability, an improvement upon Kellaris et al.'s $O(N^4 \log N)$.

1 Introduction

During the past decade, an increasing amount of organizations have started to outsource their IT infrastructure to cloud providers. This usually means that they run their web servers in the cloud, they store their data in the cloud, and they run most of their applications, including databases, in the cloud as well. Outsourcing the IT infrastructure to the cloud has several advantages: it provides reliability, availability, 27/4 support, and even low cost due to the economies of scale.

Unfortunately, outsourcing the IT infrastructure to the cloud has its drawbacks as well. For example, an organization's data may contain confidential information that should not be leaked to third parties. Storing this information outside the organization's premises is frequently discouraged, if not totally forbidden. Recent large-scale data breaches (such as that of Equifax [1]) scare organizations even further away from the cloud.

One way to deal with these restrictions and risks is to store data in the cloud in an *encrypted* form. Indeed, data leaks are no threat to encrypted data as data decryption is usually extremely difficult without the encryption key(s). Data encryption also protects the data from "curious" eyes, including the cloud provider itself. Indeed, no matter how long one "looks" at encrypted data, she will probably never be able to decrypt them without the appropriate key(s). Unfortunately, even encrypted data are not safe from curious eyes.

Indeed, previous work has demonstrated that if one monitors query results, she might be able to gain information about the data—even when stored and transmitted in an encrypted form. In particular, *range queries* (queries that return a range of scalar data) are particularly susceptible, as they have the potential to leak information about the data they access. Such information may include the *order* of the (encrypted) elements (i.e., which is larger and which is smaller) as well as the actual values of the (encrypted) elements. This latter information, essentially implies that the database can be practically decrypted.

In this paper, we focus on secure (encrypted) one-dimensional databases that support range queries on encrypted data. We assume an *honest but curious* attacker who is able to monitor all (encrypted) queries and their (encrypted) results. We develop an attack that can fully reconstruct the database after seeing enough queries. The attack first reconstructs the order of all the (encrypted) database elements and then decrypts their values.

Our attack utilizes two common types of leakage, *access pattern leakage* and *search pattern leakage*. Previous algorithms on the full database reconstruction problem depend on access pattern leakage and on a client issuing *uniformly at random queries* [8], [11], or only work on dense databases [13]. Also, some of the previous work considers additional assumptions on the database, such as the existence of points in particular intervals and/or a minimum distance between such points [8]. However, it is unlikely that a client issues queries uniformly at random. Also, not all databases are dense. Finally, special assumptions on nonempty intervals and minimum distance between points may not hold.

We have developed a *general* attack on encrypted databases that achieves full database reconstruction, recovering the exact values of all elements, after seeing all possible query results.

For the sake of timing analysis and comparison with related work, if we assume that the client issues queries uniformly at random, the algorithm successfully completes after about $O(N^2 \log N)$ queries with high probability, where N is the database size. However, our attack works no matter how the client issues queries, as long as each possible query is issued at least once.

1.1 Our focus

This paper presents an attack on encrypted databases that support range queries. We develop a general attack that achieves *full database reconstruction* (*FDR*), that is, we are able to reconstruct the exact value of all elements in the database. Our attack consists of two parts:

1. An algorithm that reconstructs the order of the elements in a database using access pattern leakage (Section 4).
2. An algorithm that achieves full database reconstruction after observing all possible queries at least once using access pattern leakage and search pattern leakage, given the order of the elements (Section 5).

Previous attacks that achieve FDR require access pattern leakage and a client that issues queries uniformly at random. We do not assume such a restrictive assumption about the client's behavior, just that she issues queries returning at least once each of the possible query results. In all cases, our algorithm improves or matches the query complexity reported in previous work, as illustrated in Table 1.

Assuming that the client issues queries uniformly at random, the algorithm successfully completes after $O(N^2 \log N)$ queries with high probability, where N is the size of the database. Our results hold with high probability, that is, probability greater than $1 - \frac{1}{N^2}$.

Table 1. Comparison of approaches to the Full Database Reconstruction (FDR) and the Full Ordering Reconstruction (FOR) problems. We compare our work with three relevant papers in the area. We report the query complexity and we highlight the best algorithm(s). In the first column, "Dense" refers to a dense database, "Any" refers to an arbitrary database, and "Any*" refers to the assumption introduced in [8] that requires the existence of points in particular intervals and/or force a minimum distance between such points. In all cases, we improve or we match the performance of previous algorithms.

	Previous Work			
Database / Problem	Kellaris et al. [11]	Lacharité et al. [13]	Grubbs et al. [8]	This Paper
Dense / FDR	$O(N^2 \log N)$	$N \log N + O(N)$		$O(N \log N)$
Any / FOR	$O(N^2 \log N)$			$O(N^2 \log N)$
Any* / FOR	$O(N^2 \log N)$		$O(N \log N)$	$O(N \log N)$
Any / FDR	$O(N^4 \log N)$		$O(N^4 \log N)$	$O(N^2 \log N)$
Any* / FDR	$O(N^4 \log N)$		$O(N^2 \log N)$	$O(N^2 \log N)$

1.2 Contributions

This paper has the following contributions:

1. We show that we can achieve FOR after $O(N^2 \log N)$ uniformly-at-random queries with high probability $(1 - 1/N^2)$ (Theorem 1).
2. We show that we can achieve FOR in dense datasets after $O(N \log N)$ uniformly-at-random queries with high probability $(1 - 3/N^3)$ (Theorem 2).
3. For datasets that have two data points in $[N/4, 3N/4]$ we show that we can achieve FOR in $O(N \log N)$ uniformly-at-random queries with high probability $(1 - 3/N^3)$ (Theorem 3).
4. We show that we can achieve FDR after $O(N^2 \log N)$ distinct queries with high probability $(1 - 1/N^2)$ (Theorem 4).

2 Model and Problem Statement

We consider a client that stores information on a database hosted by a server. A client can issue queries to the server using tokens, and the server issues responses.

Consider, for example, a collection of n records in a *database*. Each record (r, x) contains a unique identifier r in some set R, and some value $x = val(r)$ from some ordered set of integers X, on which range queries are performed, $X = [1, ..., N]$.

A range query $[a, b]$, where a, b are integers, returns the set of identifiers $M = \{r \in R : val(r) \in [a, b]\}$.

The *adversarial model* we consider is a persistent passive adversary, able to observe all communication between the client and the server.[1] The adversary aims to recover information about $val(r)$ for each $r \in R$.

A database is called dense if for all $x \in X$, there exists some r, such that $val(r) = x$.

The information learnt by the adversary depends on some scheme-dependent leakage. We examine two types of common leakage:

1. *Access Pattern Leakage:* If whenever the server responds to a query, the adversary observes the set of all matching identifiers,[2] M, we say that the scheme allows for access pattern leakage similarly to [11].
2. *Search Pattern Leakage:* If the adversary can tell whether two search tokens t_1 and t_2 that return the same set of identifiers correspond to the same range query or not, we say that the scheme allows for search pattern leakage.[3]

[1] Note that the adversary is not able to *decrypt* any encrypted information she can see.

[2] Note that any identifier r reveals no information on $val(r)$.

[3] Note that we do not assume that a token reveals the query the client issues. That is, the token does not reveal the interval $[a, b]$. We just assume that different tokens imply that the queries are different.

We assume that the adversary knows N, and the set of all identifiers.

Problem 1 (Full Database Reconstruction). Given a one-dimensional encrypted database that allows range queries, reconstruct the exact value of all elements.

Problem 2 (Full Ordering Reconstruction). Given a one-dimensional encrypted database that allows range queries, reconstruct the order of all elements.

3 Related Work

3.1 The Context

In this line of research we assume an *honest but curious* adversary. For example, this can be the cloud server. The server can easily observe all incoming and outgoing traffic and may possibly be able to draw conclusions about the values that exist in the database. We assume that the adversary is honest: it will not try to change the protocol, alter data, inject faulty information, collude with malicious users, etc. The adversary just monitors (encrypted) data.

Despite the availability of this approximate proximity information, the reader will notice that all these records (whether nearby to or far-away from each other) are still encrypted. Thus, the adversary might be able to know that *encrypted*(2) is close to *encrypted*(3), but she can not know that the values observed are actually 2 and 3: the adversary only sees *encrypted*(2) and *encrypted*(3). To be able to "break" the encryption, most of the literature makes some extra assumptions, which usually relate to the query distribution. One frequent such assumption made by several papers is that all range queries are issued uniformly at random by the client. That is, there are $N(N+1)/2$ possible queries ($[1,1], ..., [1,N]$, $[2,2], ..., [2,N], ..., [N-1,N], [N,N]$), and each one of them is issued with probability $\frac{2}{N(N+1)}$.

Our approach does not depend on the query distribution. Instead, we exploit search pattern leakage, a common leakage of such encryption schemes. This leakage allows us determine whether two search tokens correspond to the same query. For example, suppose there are 100 distinct queries that all return $\{a\}$, and 4 distinct queries that all return $\{b\}$. We can tell that the unoccupied space surrounding a is larger than the unoccupied space surrounding b.

3.2 Previous Results

In one of the seminal papers in this area, Kellaris et al. systematically studied the problem of database ordering and database reconstruction [11]. They proved that full database ordering can be done in $O(N^2 \log N)$. To do that they assume that they have all results of all possible queries. Based on the coupons collector problem, we need to observe about $O(N^2 \log N)$ queries to make sure (with high probability) that we have seen all distinct queries. They also proved that full database reconstruction can be done after having seen $O(N^4 \log N)$ queries.

Our work is slightly different than [11] in the sense that we use a different data structure which is suitable for keeping partially ordered datasets: PQ trees. As we observe more queries, the partially ordered dataset becomes almost ordered and after we observe all queries we have a fully ordered dataset (up to reflection). With respect to complexity, this paper matches the $O(N^2 \log N)$ bound of [11] to do full database ordering. However, we achieve full reconstruction after seeing only $O(N^2 \log N)$ queries, while [11] needs $O(N^4 \log N)$ queries.

This is because we are based on a different approach. In our work we are able to count the distinct queries that have been issued, while [11] is based on the statistical properties of the query distribution. Thus, in this paper we match the bound of [11] for ordering and we significantly improve the bound for full reconstruction.

More recently, Lacharité et al. focused on dense database reconstruction. Their main assumption is that there exists at least one record for each possible value in the range $[1, N]$ [13]. Using this density assumption they were able to have some very impressive results. Indeed, they achieve full database reconstruction in $O(N \log N)$ time. We are able to match this bound.

The latest work in the area from Grubbs, Lacharité, Minaud and Paterson [8] presents a global approach on how to deal with the problem. It defines a new approximate way of reconstruction: the sacrificial ϵ-Approximate Database Reconstruction (ϵ-ADR). ϵ is the error they are allowed to have in the reconstruction. That is, each element x is mapped in some position in the interval $[x - \epsilon N, x + \epsilon N]$.

Comparing our work with [8] is a bit tricky as [8] presents a full-fledged theory for *approximate* database reconstruction, while our work focuses on *exact* database reconstruction. To make a meaningful comparison we need to study Full Database Reconstruction (FDR) as a corner case of *approximate* database reconstruction when setting $\epsilon = 1/N$. In this case, it seems that Grubbs et al. achieves FDR on any database in $O(\epsilon^{-4} \log \epsilon^{-}1) = O(N^4 \log N)$ uniformly at random queries. In this paper we manage to achieve full database reconstruction in $O(N^2 \log N)$.

It is true that Grubbs et al. can match our bound and achieve FDR in $O(N^2 \log N)$. However, they can do so, only at the expense of the assumption that there exists an element in the interval $[0.2N, 0.3N]$ and that the client issues queries uniformly at random. We make no such assumption, as indeed, there may exist databases that do not have a value there - see for example the database we use for the proof of Lemma 1, and it is hard to predict the client's behavior.

Grubbs et al. are able to achieve even better bounds for full database ordering in $O(\epsilon^{-1} \log \epsilon^{-}1) = O(N \log N)$. However, they make several assumptions about the existence of elements in particular ranges as well as the distances between the elements. That is, they assume that there exist two records in range $[N/4, 3N/4]$ and that their distance is larger than $N/3$. They also require that the strict majority of all records are in the interval $[\epsilon N, N+1-\epsilon N]$. Although some databases might satisfy these assumptions some other databases might not. Both ours and [8]'s approach on ordering depend on PQ-trees.

Grubbs et al. [8] as well as [13] are able to achieve approximate database reconstruction assuming access to an auxiliary dataset distribution for the target dataset. Our work focuses on exact database reconstruction, not approximate, and thus this result is less relevant.

There have been plenty of attacks on different types of leakage as well. Kornaropoulos, Papamanthou and Tamassia [12] developed an approximate reconstruction attack utilizing leakage from k-nearest neighborhood queries. Grubbs, Lacharité, Minaud, and Paterson [7] utilize volume leakage from responses to range queries to achieve full database reconstruction. Grubbs, Ristenpart, and Shmatikov [10] present a snapshot attack that can break the claimed security guarantees of encrypted databases.

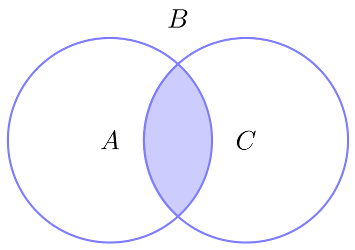
There are more attacks in the area, but those attack property-revealing-encryption schemes, which reveal more information than we assume or a more active adversary ([3], [5], [9], [6], [14], [15]).

4 Ordering Reconstruction

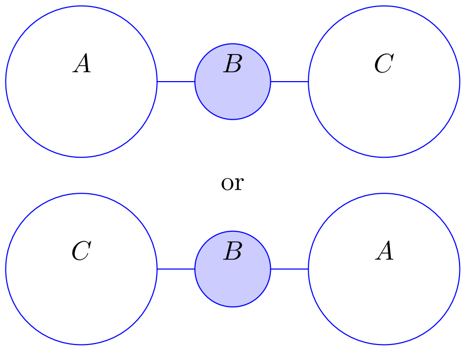
Before we attempt a full reconstruction, we shall order the records by their value.

4.1 Main Observation

The main observation of the algorithm is the following: suppose that we have two query responses which are essentially two sets of (encrypted) database elements: M_1 and M_2. Let us assume that the first set consists of two disjoint subsets (A and B): $M_1 = A \cup B$. Let us also assume that the set M_2 also consists of two disjoint subsets (B and C): $M_2 = B \cup C$. Let us also assume that A and C are also disjoint.



Then, there can be two correct (partial) orderings: (i) A, followed by B, followed by C or (ii) C, followed by B, followed by A.



The above observation can serve as a building block for ordering the elements of the database. That is, every time we see two query results that have a non-empty intersection, we know that there are two ordering possibilities: the one reflection of the other. Suppose how that we see a query result $M_3 = B \cup A_1$ and that $A_1 \cup A_2 = A$. Then, we refine the ordering as follows: A_1 followed by A_2, followed by B, followed by C, or C followed by B, followed by A_1, followed by $A2$. It seems that most query results we see have the potential to refine this partial ordering, possibly until the point where all elements will have been ordered. Although keeping and maintaining this partial ordering may seem complicated, fortunately, Booth and Lueker [2] designed a data structure that does just that: maintains a partial ordering of a set of elements. The data structure is called a PQ tree.

PQ Trees A PQ tree is a data structure that can efficiently store all permissible permutations of a set of elements.

A PQ tree contains two types of nodes, P nodes and Q nodes. It also contains leaves, which contain elements. A P node allows all possible permutations of its children[4], and a Q node allows only the order its children are in, and the reverse[5]. To use a PQ tree, one first creates a root P node that contains all elements. Then, the PQ tree can consume sets of elements that need to be contiguous and modify itself to represent these requirements. The order is fully reconstructed if the PQ tree contains one Q node, whose children are all leaves.

4.2 Algorithm

Similarly to [4] and [8], we will use PQ trees as defined in [2] to store the partial ordering of the set of database elements. The adversary initializes a PQ tree. Then, it feeds it sets of identifiers as they are observed. PQ trees can easily `update` (in linear time to the number of the records) to include newly seen sets of contiguous elements. The details of this `update` operation are beyond the scope of this paper. We are just going to use PQ trees as a module that

[4] A P node is essentially a set of elements.

[5] A Q node is like an ordered list, that represents both the list and its reverse.

operates in linear (to the number of records) time for each update operation. Note that in this work, much like all previous papers, we are not concerned with the *computational complexity* of the algorithms we use (as long as it is within reasonable polynomial bounds), but with the *number of queries* needed to achieve the database order/value reconstruction necessary. At every point the adversary has access to all allowable permutations of the identifiers using the PQ-tree.

Algorithm 1 Reconstruct Ordering of Identifiers

```
1: Initialize empty PQ-tree with the universe of possible identifiers.
2: while A new set $M$ comes do
3:     update PQ-tree with $M$.
```

4.3 Timing Analysis

Theorem 1. *Algorithm 1 can reconstruct the order of the identifiers with respect to their values after observing* $2.1N^2 \log N$ *uniformly at random issued queries, with probability greater than* $1 - 1/N^2$.

Proof. There are $N(N+1)/2$ possible queries. Given that queries come uniformly at random, the probability that a given query is not issued after $2.1N^2 \log N$ queries is

$$\left(1 - \frac{2}{N(N+1)}\right)^{2.1N^2 \log N} \leq \frac{1}{e^{4 \log N}} \leq \frac{1}{N^4}.$$

By Union Bound, the probability that at least one query is not issued after $2.1N^2 \log N$ queries is at most

$$\sum_{i=1}^{N(N+1)/2} \frac{1}{N^4} \leq \frac{N(N+1)}{2N^4} \leq \frac{1}{N^2}.$$

Thus, after $2.1N^2 \log N$ queries, all queries will have been issued with probability greater than $1 - \frac{1}{N^2}$.

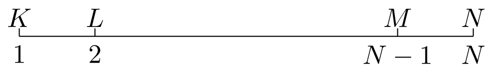
□

Note that Algorithm 1 works with any query distribution - not just with uniform ones. In the Theorem above we made the assumption that the client issues queries uniformly at random so as to be able to compare our results with the results previously reported in the literature which make this assumption.

4.4 Lower Bound of $\Omega(N^2)$ in Expectation

Lemma 1. *Let A be an adversary that can reconstruct the order of the records with only access to access pattern leakage. If the client queries ranges uniformly at random, then adversary A needs to observe at least $\Omega(N^2)$ queries before she can successfully complete the reconstruction in expectation.*

Proof. We are going to base our proof on a database that is difficult to reconstruct. Suppose we have the following database:



The only elements in it are 1,2, $N-1$ and N. That is we have one small cluster at 1,2 and one small cluster at $N-1$, and N.

Given that adversary A only has access to access pattern leakage, the possible sets A can observe are:

$$\{K\}, \{L\}, \{M\}, \{N\}$$
$$\{K, L\}, \{L, M\}, \{M, N\}$$
$$\{K, L, M\}, \{L, M, N\}$$
$$\{K, L, M, N\}$$

Given that the queries come uniformly at random, A will be able to tell that K and L are clustered together and that M and N are also clustered together relatively quickly. What drives this lower bound is that one of $\{L, M\}$, $\{K, L, M\}$, and $\{L, M, N\}$ is necessary in order to glue the two clusters together.

Note that there are $O(N^2)$ possible queries. The only query that returns $\{L, M\}$ is $[2, N-1]$, the only query that returns $\{K, L, M\}$ is $[1, N-1]$, and the query that returns $\{L, M, N\}$ is $[2, N]$.

The probability that a random query is either one of those is $\frac{3}{O(N^2)} = \frac{1}{O(N^2)}$. Thus, Adversary A has to observe at least $\Omega(N^2)$ queries to access one of the necessary results in expectation.

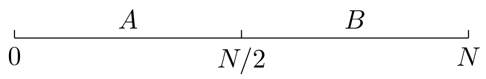
□

4.5 Dense Datasets

For dense databases, finding the order of the records' values corresponds to a full reconstruction of the database (up to reflection). In this setting, Algorithm 1 matches the best previously known complexity for dense full database reconstruction [13].

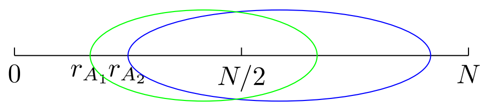
Theorem 2. *Suppose an attacker uses Algorithm 1 to reconstruct a dense database. Then, the attacker can reconstruct the database after the client issues $9.2N \log N$ uniformly at random queries with probability greater than $1 - \frac{3}{N^3}$.*

Proof. First, let's split the database in two equal parts, A and B.



By Lemma 2, after $4.1N \log N$ uniformly at random queries, for each value $a \in A$, the client issues a query $[a, b]$, for some $b \in B$ with high probability.

Let's look at the first 2 records in A, r_{A_1} and r_{A_2}. By Lemma 2, the attacker will see some response that contains $\{r_{A_1}, r_{A_2}, ...\}$, and a response that contains $\{r_{A_2}, ...\}$. Note that $\{r_{A_1}, r_{A_2}, ...\}$ contains all records in A, and $\{r_{A_2}, ...\}$ contains all elements in A besides r_{A_1}.



Given the two responses the PQ tree will be able to at least tell that r_{A_1} is to the left (or to the right) of r_{A_2} and all the other elements in A.

Similarly, given some r_{A_k}, and $r_{A_{k+1}}$, the attacker sees responses $\{r_{A_k}, r_{A_k+1}, ...\}$, and a response that contains $\{r_{A_{k+1}}, ...\}$. When she updates the PQ tree with the responses, the PQ tree will again be able to tell that r_{A_k} is to the left (or to the right) of $r_{A_{k+1}}$ and all the other elements in A higher than $r_{A_{k+1}}$.

In this way, the attacker can order all elements in A, and get

$$r_{A_1} - r_{A_2} - .... - r_{A_{max}}.$$

The attacker knows this order, but doesn't know if r_{A_1} or $r_{A_{max}}$ is the smallest element. Using a similar argument, accompanied by Lemma 3, the attacker can order all elements in B.

$$r_{B_1} - r_{B_2} - .... - r_{B_{max}}.$$

With only the above information, the PQ tree will be equivalent to one whose root will have two children P nodes. The first P node will contain the elements in A and the second P node will contain the elements in B.

It remains to show that the PQ tree can connect the two together. According to Lemma 4, the client will issue some query $[a, b]$, such that a and b are not 1 or N. As the database is dense, this query will result to a set S that contains some records from A and some records from B. Thus, the PQ tree will see that $r_{A_{max}}$ and r_{B_1} are contained in S, and thus must be next to each other. Thus, the PQ tree will return the following order:

$$r_{A_1} - r_{A_2} - .... - r_{A_{max}} - r_{B_1} - r_{B_2} - .... - r_{B_{max}}.$$

Thus, we conclude by Union Bound, that after $9.2N \log N$ queries the attacker can reconstruct the dense database with probability greater than $1 - \frac{3}{N^3}$. □

Below, we prove the Lemmas used above.

Lemma 2. *After* $4.1N \log N$ *uniformly at random queries, for each value* $a \in A$, *the client issues a query* $[a, b]$, *for some* $b \in B$ *with probability greater than* $1 - \frac{1}{N^3}$.

Proof. Let's look at one value $a \in A$. There are $N/2$ values $b \in B$. The probability that a single query issued is of the form $[a, b]$ is

$$\frac{N/2}{N(N+1)/2} = \frac{1}{N+1}.$$

After $4N \log N$ queries, the probability that no query is of the desired form is

$$\left(1 - \frac{1}{N+1}\right)^{4.1N \log N} \leq \frac{1}{e^{4 \log N}} \leq \frac{1}{N^4}.$$

Now, let's look at every $a \in A$. After $4.1N \log N$ queries, by Union Bound the probability that for at least one a the client doesn't issue a query of the form $[a, b]$ is less than

$$N \cdot \frac{1}{N^4} \leq \frac{1}{N^3}.$$

□

Lemma 3. *After* $4.1N \log N$ *uniformly at random queries, for each value* $b \in B$, *the client issues a query* $[a, b]$, *for some* $a \in A$ *with probability greater than* $1 - \frac{1}{N^3}$.

The proof follows similarly to Lemma 2.

Lemma 4. *After* $N \log N$ *uniformly at random queries, the client issues a query* $[a, b]$, *for some* $a \in A$, *and* $b \in B$, *such that* a, b *are not 1 or* N *respectively with probability greater than* $1 - \frac{1}{N^3}$.

Proof. There are N possible queries that start at 1 and N possible queries that end at N. There is one common query in these two sets ($[1, N]$). Thus, the probability that a query issued starts or ends at an extreme point is

$$\frac{2N - 1}{N(N+1)/2}.$$

Thus, the probability that after $N \log N$ issued queries the client only issued queries that start or end at an extreme point is

$$\left(\frac{2N - 1}{N(N+1)/2}\right)^{N \log N} \leq \frac{1}{N^3}.$$

□

4.6 Datasets Similar to [8]

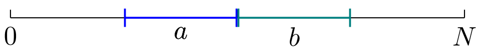
Grubbs et al. [8] assume that the database contains two records $a, b \in [N/4, 3N/4]$, such that $b - a \geq N/3$, and there are at least three records in the database, at least 1 apart.

Theorem 3 shows that Algorithm 1 matches Grubbs et al. [8] query complexity.

Theorem 3. *Suppose an attacker uses Algorithm 1 to reconstruct a database similar to the one in [8]. Then, the attacker can reconstruct the database after the client issues* $14.2N \log N$ *uniformly at random queries with probability greater than* $1 - \frac{3}{N^3}$.

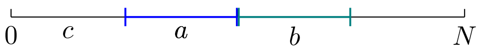
Proof. Similarly to the proof of Theorem 2, the attacker will be able to reconstruct the order of the two halves in the database after $8.2N \log N$ queries. It remains to show that she can combine them together successfully.

Like [8], we assume that the database contains two records $a, b \in [N/4, 3N/4]$, such that $b - a \geq N/3$. Thus, $a \in A$ and $b \in B$.



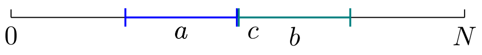
Like [8], we also assume that there is at least one more point in the database. This point c can be in one of three intervals, in $[0, val(a)]$, $[val(a), val(b)]$, or in $[val(b), N]$.

1. $c \in [0, val(a)]$



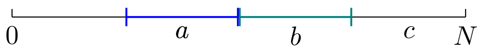
   In this case, the attacker knows that a and c are in A, and b is in B. To resolve the ordering, the attacker needs to observe set $\{a, b\}$, in order to determine that c is not between a and b.
   Note that even if a and c were right next to each other there are at least $N/4$ possible queries that return $\{a, b\}$.

2. $c \in [val(a), val(b)]$



   Without loss of generality, let's assume that the attacker knows that c is in B. In order to resolve this, the attacker has to observe some query that returns $\{a, c\}$. No matter how close a and c are, there are at least $N/4$ queries that return $\{a, c\}$. They are of the form $[x, val(c)]$, where $x \in [0, val(a)]$.

3. $c \in [val(b), N]$



This is similar to the first item. There are at least $N/4$ queries that return a query whose result is $\{a, b\}$.

In all cases above, there are at least $N/4$ queries that can resolve the ordering. The probability that none of the queries issued after $6N \log N$ queries is of the desired form is

$$\Big(1 - \frac{N/4}{N(N+1)/2}\Big)^{6N\log N} = \Big(1 - \frac{1}{2(N+1)}\Big)^{6N\log N}$$
$$\leq \frac{1}{e^{3\log N}}$$
$$\leq \frac{1}{N^3}$$

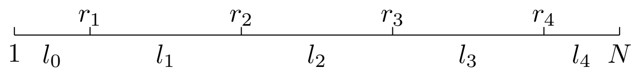
Thus, after $14.2N \log N$ queries the adversary will successfully reconstruct the order of the database with probability greater than $1 - \frac{3}{N^3}$.

□

5 Full Reconstruction

The full reconstruction will begin when the attacker has seen all possible queries. Since she knows N and we assume search pattern leakage, she will have seen all possible queries when the number of the received distinct queries is $N(N+1)/2$.

5.1 Observation



Suppose the server is hosting the above database which has records r_1, r_2, r_3, and r_4.

At this stage we assume that the attacker has already found the order of the records (up to reflection) and now is trying to determine distances l_0 (i.e. the distance between 1 and r_1), l_1 (i.e. the distance between r_1 and r_2), l_2, l_3 and l_4.

To estimate l_0 and l_1 we focus on all the possible range queries that may return r_1 and only r_1 as a response. These queries can be:

$$[1, l_0], [1, l_0 + 1], ..., [1, l_0 + l_1 - 1]$$
$$[2, l_0], [2, l_0 + 1], ..., [2, l_0 + l_1 - 1]$$
$$...$$
$$[l_0, l_0], [l_0, l_0 + 1], ..., [l_0, l_0 + l_1 - 1]$$

The number of the above queries is $l_0 \cdot l_1$. In other words, there exist exactly $l_0 \cdot l_1$ queries (different from each other) that all return the same result: r_1. Similarly we can show that there exist exactly $l_1 \cdot l_2$ possible queries (different from each other) that return a result containing only r_2, etc.

The above result can be generalized for query results returning two values. For example, there exist exactly $l_0 \cdot l_2$ different (from each other) queries that return a result containing **both** r_1 and r_2.

Once all queries have been seen, the attacker can count how many queries return each possible response. For example, let us assume that the attacker has seen exactly q_1 different queries which have returned as a result only r_1. Let us also assume that the attacker has seen exactly q_2 different queries which have returned as a result only r_2. Finally, let us also assume that the attacker has seen exactly q_{12} different queries which have returned as a result a set containing **both** r_1 and r_2.

This implies that the following equations hold:

$$\begin{aligned} l_0 \cdot l_1 &= q_1 \\ l_1 \cdot l_2 &= q_2 \\ l_0 \cdot l_2 &= q_{12} \end{aligned} \qquad \text{or} \qquad \begin{aligned} \log l_0 + \log l_1 &= \log q_1 \\ \log l_1 + \log l_2 &= \log q_2 \\ \log l_0 + \log l_2 &= \log q_{12} \end{aligned}$$

By solving the above set of equations the attacker can find the values of l_0, l_1 and l_2. Once these three have been determined, the attacker can easily find the remaining lengths l_3 and l_4 in the same spirit.

Note that search pattern leakage is instrumental for this algorithm. The attacker has to calculate q_is precisely, and she couldn't do that if she didn't know whether two tokens correspond to the same query.

5.2 Algorithm

The above example can generalize for any sets of records as follows. Let us assume that the attacker has determined the (full) ordering of the records. Let us assume that this is $r_1, r_2, ..., r_n$. Let us also assume that the number of different queries which return as a result only the record r_i is q_i. The, the attacker knows that the following set of equations hold:

$$\begin{aligned} l_0 \cdot l_1 &= q_1 \\ l_1 \cdot l_2 &= q_2 \\ &... \\ l_{n-1} \cdot l_n &= q_n \\ &\text{and} \\ l_0 \cdot l_2 &= q_{12} \end{aligned} \tag{1}$$

The above set of equations can be easily transformed into a linear system of equations.

$$\begin{aligned}\log l_0 + \log l_1 &= \log q_1\\ \log l_1 + \log l_2 &= \log q_2\\ &\dots\\ \log l_{n-1} + \log l_n &= \log q_n\\ &\text{and}\\ \log l_0 + \log l_2 &= \log q_{12}\end{aligned} \tag{2}$$

The adversary easily solve the system to find $l_0, l_1, ..., l_n$.

5.3 Pseudocode

Algorithm 2 Full Reconstruction

```
1: Run Algorithm 1
2:
3: Once all distinct queries have been received:
4: Let $order$ be the ordered list of records given by Algorithm 1
5:
6: for $i$ in range $[1, n]$ do
7:     $r = order[i]$
8:     Let $q_i$ be the number of distinct queries that returned $\{r\}$
9:     Create equation $l_{i-1} \cdot l_i = q_i$
10:
11: Let $q_{12}$ be the number of distinct queries that return $\{order[1], order[2]\}$.
12: Create equation $l_0 \cdot l_2 = q_{12}$
13:
14: Solve the system of equations
15: Return $l_i$, $i \in [0, n]$
```

5.4 Analysis

Theorem 4. *After receiving $2.1N^2 \log N$ queries issued uniformly at random, Algorithm 2 will succeed in a full reconstruction of the database with probability greater than $1 - 1/N^2$.*

Proof. Similarly to the proof of Theorem 1 we can show that after $2.1N^2 \log N$ uniformly at random issued queries with probability greater than $1 - 1/N^2$, the attacker will observe all queries at least once.

Then, the attacker can solve the (1) system of equations in page 15 to determine the distances between all record values and thus fully reconstruct the database.

6 Conclusions

During the past few years an increasing number of organizations have chosen to store their databases on the cloud. To protect the confidentiality of the data and the privacy of their users, these organizations choose to store (and process) the data in encrypted from. Unfortunately, repeated queries on the (encrypted) data may reveal information about the database records including their order and possibly their decrypted values. In this paper we show that this is possible with no more than $O(N^2 \log N)$ range queries or even as few as $O(N \log N)$ in some cases. Our work improves or matches previous bounds, while in the process making less restrictive assumptions.

References

1. Bernard, T.S., Hsu, T., Perlroth, N., Lieber, R.: Equifax says cyberattack may have affected 143 million in the u.s. https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html (2017), accessed: 2019-01-21
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. Journal of Computer and System Sciences **13**(3), 335–379 (1976)
3. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security. CCS '15, ACM, New York, NY, USA (2015)
4. Dautrich, Jr., J.L., Ravishankar, C.V.: Compromising privacy in precise query protocols. In: Proceedings of the 16th International Conference on Extending Database Technology. EDBT '13, ACM, New York, NY, USA (2013)
5. Durak, F.B., DuBuisson, T.M., Cash, D.: What else is revealed by order-revealing encryption? In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. CCS '16, ACM, New York, NY, USA (2016)
6. Grubbs, P., Sekniqi, K., Bindschaedler, V., Naveed, M., Ristenpart, T.: Leakage-abuse attacks against order-revealing encryption. In: 2017 IEEE Symposium on Security and Privacy (SP) (May 2017)
7. Grubbs, P., Lacharité, M.S., Minaud, B., Paterson, K.G.: Pump up the volume: Practical database reconstruction from volume leakage on range queries. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS '18, ACM, New York, NY, USA (2018)
8. Grubbs, P., Lacharité, M.S., Minaud, B., Paterson, K.G.: Learning to reconstruct: Statistical learning theory and encrypted database attacks. Cryptology ePrint Archive, Report 2019/011 (2019), https://eprint.iacr.org/2019/011
9. Grubbs, P., McPherson, R., Naveed, M., Ristenpart, T., Shmatikov, V.: Breaking web applications built on top of encrypted data. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. CCS '16, ACM, New York, NY, USA (2016)
10. Grubbs, P., Ristenpart, T., Shmatikov, V.: Why your encrypted database is not secure. In: Proceedings of the 16th Workshop on Hot Topics in Operating Systems. HotOS '17, ACM, New York, NY, USA (2017)

11. Kellaris, G., Kollios, G., Nissim, K., O'Neill, A.: Generic attacks on secure outsourced databases. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM (2016)
12. Kornaropoulos, E.M., Papamanthou, C., Tamassia, R.: Data recovery on encrypted databases with k-nearest neighbor query leakage. In: Proc. IEEE Symposium on Security and Privacy. pp. 245–262 (2019)
13. Lacharité, M.S., Minaud, B., Paterson, K.G.: Improved reconstruction attacks on encrypted data using range query leakage. In: 2018 IEEE Symposium on Security and Privacy (SP). IEEE (2018)
14. Pouliot, D., Wright, C.V.: The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. CCS '16, ACM, New York, NY, USA (2016)
15. Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: 25th USENIX Security Symposium (USENIX Security 16). USENIX Association, Austin, TX (2016)