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 data in the cloud, a database can be stored in encrypted form and all queries can be executed on 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. We consider an encrypted database whose records have values from a given universe of N consecutive integers. Our attack assumes access pattern and search pattern leakage. It succeeds after the attacker has seen each of the possible query results at least once, independent of their distribution. If we assume that the client issues queries uniformly at random, we can decrypt *the entire* database with high probability after observing $O(N^2 \log N)$ queries.

1 Introduction

During the past decade, an increasing number of organizations have started to outsource their computing infrastructure to cloud providers. This usually means that they store their data in the cloud and run most of their applications, including databases, in the cloud as well. Outsourcing data storage and computation to the cloud has several advantages, including reliability, availability, and 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 unauthorized third parties. Storing this information outside the organization's premises may be challenging, and in some cases unfeasible, due to a multitude of business and regulatory constraints.

One way to deal with these restrictions and risks is to store data in the cloud in *encrypted* form. Indeed, data leaks are no threat to encrypted data as decryption is unfeasible without possession of the key. Besides malicious attacks, encryption also protects data from "curious" eyes, including the cloud provider itself.

Unfortunately, even encrypted data is not safe from curious eyes when searched upon. Indeed, previous work has demonstrated that an attacker who monitors query results might be able to gain information about the data—even when stored and transmitted in encrypted form. In particular, *range queries* (queries that return database records with attribute values in a given interval) 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) records (i.e., which has larger and which has smaller value) as well as the actual values of the (encrypted) records. This latter information essentially implies that the database can be practically decrypted.

In this paper, we focus on 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 reconstructs 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 *queries from a known distribution* [8, 14], or only work on dense databases [18]. 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 in practical applications. 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.

1.1 Organization of the Paper

This paper presents an attack on encrypted databases that support range queries. We assume that the attacker has observed all possible queries at least once. We exploit access pattern leakage to achieve *full ordering reconstruction* (*FOR*), that is, reconstruct the order of the database elements induced by their values (but not the values themselves). Next, we exploit both access pattern leakage and search pattern leakage to achieve *full database reconstruction* (*FDR*), that is, we are able to reconstruct the exact value of all elements in the database.

After defining our model (Section 2) and reviewing related work (Section 3), we present our algorithm for full ordering reconstruction in Section 4 and our algorithm for full database reconstruction in Section 5. Finally, Section 6 concludes the paper outlining directions for future work.

1.2 Contributions

Previous attacks that achieve full database reconstruction (FDR) use access pattern leakage and a client that issues queries according to a known distribution. Our attack makes a different assumption. We do not assume that the client issues queries in any particular distribution, but we do assume that the searchable encryption scheme leaks the search pattern. Notably, typical searchable encryption schemes do leak the search pattern.

We provide in Table 1 a comparison of our work with selected papers [6, 8, 14, 18] on full database reconstruction (FDR) and full ordering reconstruction (FOR) from range queries on one-dimensional encrypted databases. For each approach, the table shows the assumptions on the leakage observed by the attacker and knowledge of the query distribution by the attacker. Also, for the scenario of a client that issues queries uniformly at random (a standard scenario in the literature), the table shows the query complexity of FDR and FOR attacks on various types of databases. As shown in the table, our method improves or matches the query complexity reported in previous work, albeit under different assumptions.

Table 1. Comparison of approaches to full database reconstruction (FDR) and full ordering reconstruction (FOR) from range queries on one-dimensional encrypted databases. We compare our work with three relevant papers in the area by showing the assumptions on the attacker's capabilities and the query complexity of the attack for the case of a client that issues queries uniformly at random, highlighting the best asymptotic bounds. The query complexity is expressed in terms of the size of the universe of database elements, N. The following types of databases are considered: "Any" refers to an arbitrary database, "Dense" refers to a dense database, which has at least one record for each possible value, and "Any*" refers to the assumption introduced in [6, 8] that requires the existence of values in particular intervals and/or forces a minimum distance between such points.

	Previous Work			**This Paper**
	Kellaris et al. [14]	Lacharité et al. [18]	Grubbs et al. [6, 8]	
Assumptions				
Access Pattern Leakage	✓	✓	✓	✓
Search Pattern Leakage				✓(only FDR)
Known Distribution	✓(only FDR)		✓(only FDR)	
Database / Problem				
Dense / FDR	$O(N^2 \log N)$	$N \log N + O(N)$	$O(N \log N)$	$O(N \log N)$
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	$O(N^4 \log N)$		$O(N^2 \log N)$	$O(N^2 \log N)$

The main contributions of this paper are summarized as follows, where N denotes the size of the universe of database elements:

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 a dense database 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 $[N/4, 3N/4]$ and their distance is larger than $N/3$, we show that we can achieve FOR after $O(N \log N)$ 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).

Kellaris et al. [14] have shown that there exist datasets which cannot be distinguished by attackers that observe significantly fewer than $O(N^4)$ queries chosen uniformly at random. However this lower bound works for attacks that use access pattern or communication volume leakage. We use an additional type of leakage, search pattern leakage, which allows us to achieve faster attacks.

2 Model and Problem Statement

We consider a client that stores information on an encrypted database hosted by a server. The client issues range queries to the server using tokens, and the server returns responses to the queries.

We define a *database* as a collection of n records, where each record (r, x) comprises a unique *identifier*, r, from some set R, and a *value* $x = val(r)$ from an interval of integers $X = [1, ..., N]$, which is the universe of database values. A database is called *dense* if for all $x \in X$, the database contains some record (r, x) such that $val(r) = x$. Note that there may be multiple records with the same value. A *range query* $[a, b]$, where $a \leq b$ 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 who is able to observe communication between the client and the server. The adversary aims to recover value $val(r)$ for each identifier, r, in the database. Note that the adversary is not able to *decrypt* any observed encrypted data. The information learnt by the adversary depends on some scheme-dependent leakage.

We examine two types of common leakage:

– *Access Pattern Leakage:* If whenever the server responds to a query, the adversary observes the set of all matching identifiers, M, we say that the scheme allows for *access pattern leakage*. We assume that the identifier r reveals no information on $val(r)$.
– *Search Pattern Leakage:* If the adversary can observe search tokens and determine whether two tokens, t_1 and t_2, correspond to the same range query, we say that the scheme allows for *search pattern leakage*. Note that we do not assume that a token reveals the query the client issues. That is, the token

does not indicate the range $[a, b]$. We just assume that the adversary can distinguish whether two query ranges are the same or different by observing the corresponding tokens.

In this paper, we consider the following two problems and present efficient algorithms for them.

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

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

Our algorithms assume that the adversary knows the size of the universe of database values, N. Our FOR algorithm, presented in Section 4, assumes access pattern leakage while our FDR algorithm, presented in Section 5, assumes both access pattern leakage and search pattern leakage.

3 Related Work

3.1 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: she will not try to change the protocol, alter data, inject faulty information, collude with malicious users, etc. The adversary just monitors (encrypted) data.

Given that data are stored in an encrypted form, one might be tempted to think that it is not possible to decrypt them unless the decryption key can be found. Unfortunately, this is not the case. If the database supports range queries, an adversary who monitors the traffic is able to find *some* information about the records observed. For example, one piece of information that can be easily found is that *all the results of a range query belong in the same range* (by definition) and are, in one way or another, "close" to each other. By observing queries for a very long time, one might be able to infer which records are likely to be in proximity of each other (e.g., those that frequently occur together in query results) and which records are likely to be more distant from each other (e.g., those which do not frequently occur together in query results).

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 as 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 $\frac{2}{N(N+1)}$. Note that even though all queries are issued with the same probability, some elements are queried more than others. Specifically, elements close to the middle of the database are queried more than elements towards the endpoints.

Our approach does not depend on the query distribution. Instead, we exploit search pattern leakage, a common leakage of searchable 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 the following review of previous work in the area, we denote with N is the size of the universe (interval) of database values.

A seminal paper by Kellaris, Kollios, Nissim, and O’Neill [14] is the first systematic study of the problems of full ordering reconstruction and full database reconstruction from range queries. They prove that full database ordering can be done with $O(N^2 \log N)$ queries. This attack assumes that the adversary observes the answers to all possible queries. Thus, based on the coupon collector problem, the assumption holds with high probability after $O(N^2 \log N)$ queries. They also show that full database reconstruction can be done with high probability after observing $O(N^4 \log N)$ queries. Our work differs from [14] in the use of data structures, Namely, we maintain the partial order of observed identifiers in a PQ tree [1]. As we observe more queries, we gain more information about the ordering of the identifiers, which is efficiently maintained in the PQ-tree. Eventually, once we observe all queries, we have a fully ordered set (up to reflection). With respect to query complexity, for full database ordering, we match the $O(N^2 \log N)$ bound of [14]. Also, we achieve full reconstruction after seeing $O(N^2 \log N)$ queries, while the approach by [14] needs $\Omega(N^4 \log N)$ queries. We obtain this improvement thanks to our assumption of search pattern leakage, which allows us to count the distinct queries that have been issued, while the method of [14] is based on the statistical properties of the query distribution.

Lacharité, Minaud and Paterson [18] focus on the reconstruction of a dense database, i.e., a database for which there exists at least one record for each possible value in the universe of values, $[1,N]$. Using this density assumption, they achieve an impressive speedup in the query complexity of the attack. Indeed, they achieve full database reconstruction from access pattern leakage after observing $O(N \log N)$ uniformly at random queries. We are able to match this bound by using a datastructure called a PQ tree[1]. Note that neither their method nor ours assumes knowledge of the query distribution by the adversary.

The recent work by Grubbs, Lacharité, Minaud and Paterson [6, 8] presents a comprehensive approach to database reconstruction. They generalize the problem by introducing a new approximate way of reconstruction, called ϵ-approximate database reconstruction (ϵ-ADR). In this model, ϵ is the error the attack is allowed to have in the reconstruction. That is, for each original value x, the reconstructed value is in the interval $[x-\epsilon N, x+\epsilon N]$. Note that full database reconstruction (FDR) is the special case of ϵ-ADR achieved by setting $\epsilon = 1/N$. Regarding data structures, our use of PQ-trees is similar to theirs. To compare our FDR attack to theirs, we set the approximation parameter ϵ in their ϵ-ADR model equal to $1/N$ and consider the standard scenario of queries issued uniformly at random. They achieve FDR on an arbitrary database with $O(N^4 \log N)$ queries using access pattern leakage. Instead, we obtain FDR with $O(N^2 \log N)$ queries using both access pattern leakage and search pattern leakage. They further achieve FDR with $O(N^2 \log N)$ queries under the additional assumption that the database has a record with value in the interval $[0.2N, 0.3N]$.

Regarding ordering reconstruction, they are able to achieve FOR with $O(N \log N)$ queries under the following additional assumption on the database values: there are two values in range $[N/4, 3N/4]$ and their distance is larger than $N/3$. Note that this implies that FDR can also be achieved in dense databases with $O(N \log N)$ queries.

Note that Grubbs et al. [6, 8] as well as Lacharité et al. [18] are also able to achieve approximate database reconstruction assuming access to an auxiliary distribution for the database values. 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 [15] 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. While most of the above attack papers assume that the client issues queries uniformly at random, in recent work, Kornaropoulos, Papamanthou and Tamassia [16, 17] develop distribution-agnostic reconstruction attacks from range and k-nearest neighbor (k-NN) queries using search pattern leakage.

There are also attacks on property-revealing-encryption schemes (which reveal more information than we assume) and attacks that assume a more active adversary [2, 5, 9, 11, 19, 20].

4 Full Ordering Reconstruction

In this section, we present our algorithm for full ordering reconstruction, which infers the order of the database records by value. The algorithm uses access pattern leakage, but not search pattern leakage.

4.1 Approach

The ordering reconstruction algorithm is based on the following observation. Suppose we have two query responses, M_1 and M_2, each consisting of the set of identifiers of a query response. Let $B = M_1 \cap M_2$, $A = M_1 - B$, and $C = M_2 - B$. We have $M_1 = A \cup B$ and $M_2 = B \cup C$, where A and C are disjoint. as shown in Figure 1.

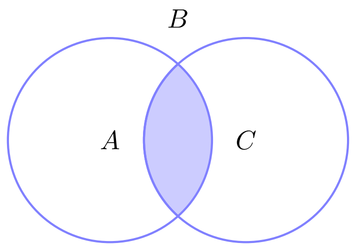


Fig. 1. Intersection and differences between two range query responses, M_1 and M_2, where $B = M_1 \cap M_2$, $A = M_1 - B$, and $C = M_2 - B$.

Then, there can be only two correct (partial) orderings of the elements in M_1 and M_2 by value: (i) A, followed by B, followed by C or (ii) C, followed by B, followed by A, as illustrated in Figure 2.

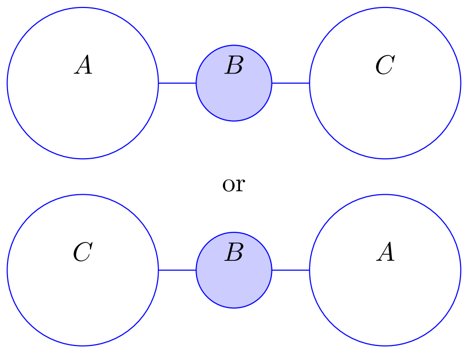


Fig. 2. The two possible ordering of the elements in the query responses $M_1 = A \cup B$ and $M_2 = B \cup C$ from Figure 1.

The above observation serves as a building block of our algorithm 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 now 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 [1] 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 is built from three types of nodes, P nodes, Q nodes, and leaf nodes:

1. *Leaf node.* A leaf node stores a single element and has no children.
2. P *node.* The children of a P node can assume any ordering. (Similar to a set.)
3. Q *node.* The children of a Q node can assume only the ordering they are in, or the reverse order. (Similar to an ordered list.)

To use a PQ tree, one first creates a root P node that contains all elements as children leaf nodes. Then, the PQ tree can consume sets of elements that need to be contiguous and modify itself to represent these requirements, by reorganizing the leaf nodes in P and Q nodes as appropriate. The order is fully reconstructed if the PQ tree contains one Q node, whose children are all leaves.

Each range response is essentially a set of identifiers that are all contiguous. The PQ tree can consume these range responses to identify all permissible permutations of the ordering of the identifiers.

4.2 Algorithm

We show in Algorithm 1 our method for full ordering reconstruction. Similarly to [3] and [8], we use PQ trees [1] 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 answers to queries are observed. For each answer set, M, the PQ tree updates the partial order of the identifiers seen so far in time proportional to the size of M by means of operation update(M) The details of operation update can be found in [1].

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 Full Ordering Reconstruction

```
1: Initialize an empty PQ-tree, $T$
2: while a new answer set $M$ is observed do
3:     $T$.update($M$)
```

4.3 Query Complexity Analysis

The query complexity of our FOR algorithm is summarized in the following theorem.

Theorem 1. *Using access pattern leakage, Algorithm 1 reconstructs the order of the database identifiers with respect to their values after observing $2.1N^2 \log N$ uniformly at random issued queries, with probability greater than $1-1/N^2$, where N is the size of the universe of database values.*

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

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

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 $1 - \frac{1}{N^2}$.

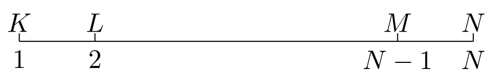
□

Note that Algorithm 1 works with any query distribution—not just with uniform ones. In the theorem above, we have 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

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 $\Omega(N^2)$ queries before successfully completing 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 element values 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.

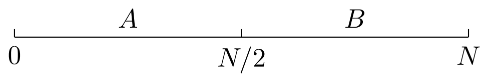
□

4.5 Dense Databases

For dense databases, reconstructing the ordering of the elements 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 [6, 8, 18].

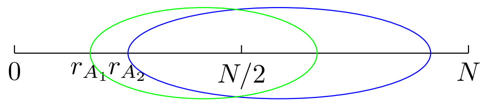
Theorem 2. *Suppose an attacker uses Algorithm 1 to reconstruct a dense database. Then, the attacker can reconstruct the database after the client issues* $8.2N\log N + 4\log N$ *uniformly at random queries with probability greater than* $1-\frac{3}{N^3}$*, where* N *is the size of the database.*

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 $[a, b]$, which is not of the form $[N/2 - i, N/2 + i + 1]$, and starts in $A - \{1\}$ and ends in $B - \{N\}$. As the database is dense, this query will result to a set S that contains some records from A and some records from B. Importantly, this query doesn't query 1 or N, and breaks the symmetry if all other queries were of the form $[N/2-i, N/2+i+1]$. Because the query returns only a subset of A and a subset of B, the PQ tree is able to deduce 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 $8.2N \log N + 4 \log N$ 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 $4.1N\log N$ 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 $4\log N$ uniformly at random queries, the client issues a query $[a, b]$ that is not of the form $[N/2-i, N/2+i+1]$, for some $a \in A-\{1\}$, $b \in B-\{N\}$, and $i \in [1, N/2)$, with probability greater than $1-\frac{1}{N^3}$.*

Proof. There are $N/2-1$ desirable queries that start on each $a \in A-\{1\}$, as one of them is not of the desired form. Thus, there are $(\frac{N}{2}-1)(\frac{N}{2}-1)$ queries of desired form.

Thus, the probability that a query issued is not of the form $[N/2-i, N/2+i+1]$ is

$$\frac{(\frac{N}{2}-1)(\frac{N}{2}-1)}{\frac{N(N+1)}{2}} \leq \frac{1}{4},$$

for $N > 8$. Thus, the probability that after $3\log N$ issued queries the client only issued undesirable queries is less than

$$\left(\frac{1}{4}\right)^{4\log N} \leq \frac{1}{N^3}.$$

□

4.6 Databases with Special Properties

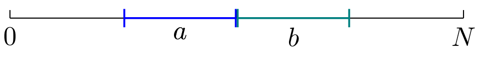
Grubbs et al. [8] assume that the database contains two records $a, b \in [N/4, 3N/4]$, such that $b-a \geq N/3$, and there are at least three records in the database, at least 1 apart.

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 $14.4N\log N$ uniformly at random queries with probability greater than $1-\frac{3}{N^3}$, where N is the size of the database.*

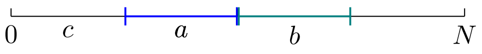
Proof. Similarly to the proof of Theorem 2, the attacker will be able to reconstruct the order of the two halves in the database after $8.2N \log N$ queries. It remains to show that she can combine them together successfully.

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 $b \in B$.



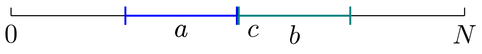
Like [8], we also assume that there is at least one more point in the database. This point c can be in one of three intervals, in $[0, val(a)]$, $[val(a), val(b)]$, or in $[val(b), N]$.

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



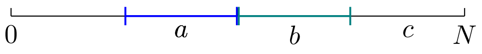
   In this case, the attacker knows that a and c are in A, and b is in B. To resolve the ordering, the attacker needs to observe set $\{a, b\}$, in order to determine that c is not between a and b.
   Note that even if a and c were right next to each other there are at least $N/4$ possible queries that 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 $6.2N\log N$ queries is of the desired form is

$$\begin{aligned}\left(1-\frac{N/4}{N(N+1)/2}\right)^{6.2N\log N} &= \left(1-\frac{1}{2(N+1)}\right)^{6.2N\log N}\\ &\leq \frac{1}{e^{3.1\log N}}\\ &\leq \frac{1}{N^3}\end{aligned}$$

Thus, after $14.4N\log N$ queries the adversary will successfully reconstruct the order of the database with probability greater than $1-\frac{3}{N^3}$.

□

5 Full Database Reconstruction

In this section, we present our algorithm for full database reconstruction, which infers the values of the database records. The algorithm uses both access pattern leakage and search pattern leakage. It assumes that Algorithm 1 for full ordering reconstruction has been already executed, hence the attacker knows the ordering of the n database records by value, $r_1, r_2, \cdots, r_n$. By using search pattern leakage, the attacker counts the number of distinct queries observed until this count reaches $N(N+1)/2$, where N is the size of the universe of database elements. This occurs when the attacker has seen all possible queries.

5.1 Example

Suppose the server is hosting a database with records r_1, r_2, r_3, and r_4, as shown in Figure 3.

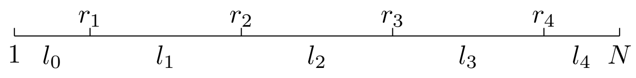


Fig. 3. Example of a database with four records with values in the interal $[1, N]$.

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 the distances between consecutive records, denoted
l_0 (distance between 1 and r_1),
l_1 (distance between r_1 and r_2),
l_2 (distance between r_2 and r_3),
l_3 (distance between r_3 and r_4), and
l_4 (distance between r_4 and N).

To determine l_0 and l_1 we focus on all the possible range queries that return (only) r_1 as a response. These queries are as follows:

$$\begin{gathered}[1, l_0], [1, l_0+1], \ldots, [1, l_0+l_1-1] \\ [2, l_0], [2, l_0+1], \ldots, [2, l_0+l_1-1] \\ \cdots \\ [l_0, l_0], [l_0, l_0+1], \ldots, [l_0, l_0+l_1-1]\end{gathered}$$

The number of the above queries is $l_0 \cdot l_1$. In other words, there exist exactly $l_0 \cdot l_1$ distinct queries that all return r_1 as a response. Similarly, we can show that there exist exactly $l_1 \cdot l_2$ distinct queries that return r_2 as a response, and so on.

The above result can be generalized for query answers comprising two records. For example, there exist exactly $l_0 \cdot l_2$ distinct queries that return the pair $\{r_1, r_2\}$ as a response.

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:

$$\begin{aligned} l_0 \cdot l_1 &= q_1 \\ l_1 \cdot l_2 &= q_2 \\ l_0 \cdot l_2 &= q_{12} \end{aligned}$$

By solving the above set of three equations, the attacker can find the values of distances l_0, l_1 and l_2. Once these three have been determined, the attacker can easily compute the remaining distances l_3 and l_4 in a similar way, thus achieving full reconstruction of the database values.

Note that search pattern leakage is instrumental for this algorithm. The attacker has to calculate the query counts (i.e., the q_i constant terms in the system of equations) precisely and can do so only by determining whether two tokens correspond to the same query.

5.2 Algorithm

The above example generalizes to any number of database records as follows. Let us assume that the attacker has determined the full ordering of the records of a database of size n, denoted $r_1, r_2, \ldots, r_n$. Let us also assume that the number of distinct queries which return as a result only record r_i is q_i and the the number of distinct queries which return as a result only the pair or records $\{r_1, r_2\}$ is q_{12}.

The attacker builds the following system of $n+1$ equations over variables l_i, $i = 0, \ldots, n$.

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

In the above system, the meaning of the variables is as follows:

– l_0 denotes the distance between 1 and r_1;
– for $i = 1, \ldots, n-1$, l_i denotes the distance between records r_i and r_{i+1}; and
– l_n denotes the distance between r_n and N.

One way to solve this system of equations is to first solve the subsystem

$$l_0 \cdot l_1 = q_1,\ l_1 \cdot l_2 = q_2, \text{ and } l_0 \cdot l_2 = q_{12}$$

for l_0, l_1 and l_2. Then, since the remaining equations are of the form $l_i \cdot l_{i+1} = q_{i+1}$, for $i \geq 2$, one can just solve for l_{i+1} one by one using the recovered values, starting with $i = 2$.

The resulting method for full database reconstruction is shown in Algorithm 2.

Algorithm 2 Full Reconstruction

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

5.3 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 $1 - 1/N^2$, where N is the size of the universe of database values.*

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 system of equations (1) to determine the distances between all record values and thus fully reconstruct the database. □

6 Conclusions

In this paper, we have presented an attack that reconstructs the values of an encrypted database from access pattern leakage and search pattern leakage. As in previous constructions, complete exact reconstruction requires observing a large number of queries and bounds on the query complexity of the attack are proved for a uniform query distribution. Recently, a reconstruction method has been presented whose efficiency does not rely on assumptions about the query distribution [17], which opens an interesting research direction. Also, in response to attack papers on searchable encryption, there is an interesting body of work that focuses on leakage reduction (e.g., [4, 12, 13]). Another promising avenue for future work is developing methods to attack the above improved schemes.

7 Acknowledgments

We would like to thank Thibaut Bagory, Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson and for their helpful comments and suggestions on a previous version of this paper.

References

[1] 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)

[2] Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: Proc. ACM Conf. on Computer and Communications Security, CCS (2015)

[3] Dautrich, Jr., J.L., Ravishankar, C.V.: Compromising privacy in precise query protocols. In: Proc. Int. Conf. on Extending Database Technology, EDBT (2013)

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

[5] Durak, F.B., DuBuisson, T.M., Cash, D.: What else is revealed by order-revealing encryption? In: Proc. ACM Conf. on Computer and Communications Security, CCS (2016)

[6] Grubbs, P., Lacharité, M., Minaud, B., Paterson, K.G.: Learning to reconstruct: Statistical learning theory and encrypted database attacks. In: IEEE Symp. on Security and Privacy, pp. 513–529 (2019)

[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: Proc. ACM Conf. on Computer and Communications Security, CCS (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: Proc. ACM Conf. on Computer and Communications Security, CCS (2016)

[10] Grubbs, P., Ristenpart, T., Shmatikov, V.: Why your encrypted database is not secure. In: Proc. Workshop on Hot Topics in Operating Systems, HotOS (2017)

[11] Grubbs, P., Sekniqi, K., Bindschaedler, V., Naveed, M., Ristenpart, T.: Leakage-abuse attacks against order-revealing encryption. In: 2017 IEEE Symp. on Security and Privacy, SP (2017)

[12] Kamara, S., Moataz, T.: Computationally volume-hiding structured encryption. In: Advances in Cryptology, EUROCRYPT (2019)

[13] Kamara, S., Moataz, T., Ohrimenko, O.: Structured encryption and leakage suppression. In: Advances in Cryptology, CRYPTO (2018)

[14] Kellaris, G., Kollios, G., Nissim, K., O'Neill, A.: Generic attacks on secure outsourced databases. In: Proc. ACM Conf. on Computer and Communications Security, ACM (2016)

[15] Kornaropoulos, E.M., Papamanthou, C., Tamassia, R.: Data recovery on encrypted databases with k-nearest neighbor query leakage. In: Proc. IEEE Symp. on Security and Privacy, pp. 245–262, SP (2019)

[16] Kornaropoulos, E.M., Papamanthou, C., Tamassia, R.: The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. Cryptology ePrint Archive, Report 2019/441 (2019), https://eprint.iacr.org/2019/441

[17] Kornaropoulos, E.M., Papamanthou, C., Tamassia, R.: The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. In: Proc. IEEE Symp.on Security and Privacy, SP (2020), to appear

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

[19] Pouliot, D., Wright, C.V.: The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In: Proc. ACM Conf. on Computer and Communications Security, CCS (2016)

[20] Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: Proc. USENIX Security Symposium (2016)