Mitigation Techniques for Attacks on 1-Dimensional Databases that Support Range Queries

Evangelia Anna Markatou and Roberto Tamassia

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

Abstract. In recent years, a number of attacks have been developed that can reconstruct encrypted one-dimensional databases that support range queries under the persistent passive adversary model. These attacks allow an (honest but curious) adversary (such as the cloud provider) to find the order of the elements in the database and, in some cases, to even *reconstruct* the database itself.

In this paper we present two mitigation techniques to make it harder for the adversary to reconstruct the database. The first technique makes it impossible for an adversary to reconstruct the values stored in the database with an error smaller than k, for k chosen by the client. By fine-tuning k, the user can increase the adversary's error at will.

The second technique is targeted towards adversaries who have managed to learn the distribution of the queries issued. Such adversaries may be able to reconstruct most of the database after seeing a very small (i.e. poly-logarithmic) number of queries. To neutralize such adversaries, our technique turns the database to a circular buffer. All known techniques that exploit knowledge of distribution fail, and no technique can determine which record is first (or last) based on access pattern leakage.

Keywords: Searchable Encryption · Encrypted Databases · Leakage-Abuse Attacks · Mitigation

1 Introduction

Currently, many organizations outsource their data, and sometimes their entire information technology infrastructure to the cloud. This is a reasonable choice as the cloud is a reliable, inexpensive, and generally safe place to store an organization's data. However, although storing data on the cloud usually provides protection from outside attackers, it may expose the data to the prying eyes of curious insiders within the provider. Thus, an additional security measure is to store encrypted data on the cloud and to support queries on the data using searchable encryption. This additional encryption step ensures that the data can not be seen in plaintext by a curious cloud provider.

1.1 Reconstruction Attacks on Encrypted Databases

Although, at a first glance, searchable encryption appears to guarantee the confidentiality of the data, unfortunately, this is not the case. Recent papers have demonstrated several attacks against encrypted databases that allow for range queries on the data. These attacks leak various amounts of information about the data, in some cases even achieving full database reconstruction.

For example, consider an encrypted one-dimensional database whose values are from a set of N consecutive integers. Kellaris, Kollios, Nissim, and O'Neill [24] have demonstrated that they can achieve full database reconstruction after observing the (encrypted) answers to $O(N^4 \log N)$ range queries. More recently, Grubbs, Lacharité, Minaud, and Paterson [20] have improved in this result and have shown how to achieve full database reconstruction from $O(N^2 \log N)$ queries. To make matters worse, they have also shown that approximate database reconstruction (i.e., reconstruction of most database elements with an asymptotically small error) can be done from a poly-logarithmic number of queries. Although approximate database reconstruction recovers only a portion of the database, this portion could be large enough for the attacker and may reveal sensitive data stored in the database.

The literature presents plenty of attacks on other various types of leakage as well. For example, Kornaropoulos, Papamanthou and Tamassia [25] have developed an approximate reconstruction attack utilizing leakage from k-nearest neighbor queries. Grubbs, Lacharité, Minaud, and Paterson [19] utilize volume leakage from responses to range queries to achieve full database reconstruction. Grubbs, Ristenpart, and Shmatikov [21] present a snapshot attack that can break the claimed security guarantees of encrypted databases.

All this previous research suggests that even if the database is stored encrypted, database reconstruction may be possible in reasonable time. Having realized that current searchable encryption approaches provide little protection against powerful attackers such as *honest but curious cloud providers*, in this paper we take a slightly different approach and explore whether it is possible to make the task of the attacker a bit more difficult by introducing some form of *noise* or some kind of error by *changing* the queries issued by the clients. For example, when a client issues query $[a, b]$, our methods issue query $[a', b']$ (where $a \neq a'$ and/or $b \neq b'$) or our methods issue multiple queries. The choice of a' and b' is done in such as way so as to *obfuscate* the real query that the clients want to issue and thus *confuse* the attacker. So far, all attacks either assume that the client issues queries uniformly at random [20, 24] or that the adversary has access to all possible query responses [19, 28].

1.2 Contributions

After defining our encrypted database model (Section 2) and reviewing related work (Section 3), we present two *obfuscation* techniques to mitigate reconstruction attacks on encrypted databases from the observation of the answers to one-dimensional range queries. These techniques, which we call *blocked queries* and *wrap-around queries*, are summarized below.

– **Blocked Queries or BQ (Section 4).**
Our first technique modifies the queries issued by the client. The client selects an integer parameter k. When the client wants to issue a query $[a, b]$, BQ rounds a down to the nearest smaller multiple of k, and rounds b up to the nearest larger multiple of k minus 1. That is, the client issues modified query:

$$[k \cdot \lfloor a/k \rfloor , k \cdot \lceil (b+1)/k \rceil - 1] .$$

In this way, the adversary can only *approximately* reconstruct the database. Indeed, we will show that for each record r_i which corresponds to value v_i, the adversary can estimate v_i only up to an error of k.
The client can pick k as they desire. Note that k introduces a trade-off between communication complexity and security: the higher the value of k, the higher the error in the adversary's approximation, but also the larger the range being queried, which can increase the communication complexity.

– **Wrap-around Queries or WQ (Section 5).**
Our second approach focuses on the scenario where the adversary knows the distribution of the queries. Indeed, a large body of previous work on attacks (including [20] and [24]) assumes that clients issue queries *uniformly at random* (or that the query distribution is known). Using this assumption, previous work managed to achieve (almost) full database reconstruction in poly-logarithmic time with only a very small amount of error of $O(1/\log N)$. Therefore, protecting the privacy of the data in settings when the adversary knows the distribution of the queries is of paramount importance.
To obfuscate the database results, when the query distribution is known, we use a four-pronged approach:
  - We introduce the notion of *wrap-around* queries. In normal queries $[a, b]$ we always assume that $a \leq b$. In contrary, in *wrap-around* queries $[a, b]$ we assume that $a > b$.
    In such cases the result of a *wrap around* query $[a, b]$ is the union of the results of normal queries $[a, N]$ and $[1, b]$. That is, a *wrap-around* query, as the name goes, *wraps around* the end of the value range (i.e. N) and continues from the beginning (i.e. 1).
    One may imagine that wrap-around queries treat the data not as a vector (from 1 to N) but as a cyclic buffer. The size of the buffer is still N, but the start of the cyclic buffer is not known. Actually, in a cyclic buffer, much like in a circle, there is no start (or end for that matter).
  - Approximately each time a client issues query $[a, b]$, WQ issues a second query: $[a', b']$.[1] The purpose of this second query is to confuse the adversary who will not be able to say whether the original query were $[a, b]$ or $[a', b']$. Note, that query $[a', b']$ has to be a bit more sophisticated. Indeed, queries $[a', b']$ are taken from a suitable distribution so that when one combines all queries $[a, b]$ and $[a', b']$, the probability of each value $v_i \in [1, N]$ being queried is the same for all i. In this way, no value v_i

[1] Depending on the distribution, WQ may need to issue several queries. For the purposes of discussion, at this point we assume that just one extra query $[a', b']$ is issued.

is more popular than the other values, removing asymmetries previously exploited by adversaries.[2]

- Range queries $[a, b]$ are issued as singleton queries $[a, a]$, $[a+1, a+1]$, $[a+2, a+2]$, ... $[b, b]$. In this way the attacker will not be able to distinguish normal queries (which would have been issued as a single query) from wrapped-around ones (which would otherwise have been issued as two queries: $[a, N]$ and $[1, b]$).
- We always issue range queries in pairs. We deconstruct all range queries into singleton queries as above, and shuffle the singleton queries of each range query. We then issue all the singleton queries to the server. [3]

Approach BQ aims to introduce an error in the reconstruction of any database. This makes exact reconstruction impossible for any adversary exploiting access pattern leakage. We note that method WQ aims to render unusable a large number of current attacks on encrypted databases, that assume that they know the query distribution. Indeed, using WQ, we break the main assumption that attacks have made so far: The assumption that when queries are issued uniformly at random, some database values appear more frequently than others.

Kellaris et al. [24] and Grubbs et al. [20] present attacks that inherently depend on the client issuing queries following some distribution. These attacks no longer work when WQ is deployed.

2 Model

We consider a client that stores information on a database hosted by a server. The client issues one-dimensional range queries to the server using tokens, and the server returns responses to such queries. Specifically, we consider a database consisting a collection of n records, where each record (r, x) comprises a unique identifier, r, from some set R, and an integer value, x, from some interval of N consecutive integers (e.g., $X = [0, \ldots, N-1]$ or $X = [1, \ldots, N]$). For a record (r, x) of the database, we use the notation $x = val(r)$ to indicate the value x associated with identifier r.

We now introduce "normal" queries, which the user can issue, and "wrap-around" queries, which they cannot.

Definition 1. (Normal Query) *A range query $[a, b]$ such that $a \leq b$, is called a* normal query. *It returns the set of all matching identifiers, $M = \{r \in R : val(r) \in [a, b]\}$.*

Definition 2. (Wrap-around Query) *A range query $[a, b]$, where $a > b$, is called a* wrap-around query. *It returns the set of all matching identifiers, $M = \{r \in R : val(r) \geq a \text{ or } val(r) \leq b\}$.*

[2] Actually, the same should be true for all combinations of values v_i and v_j as we will later show.

[3] If we issued queries one by one, the last query issued would always be a normal query.

The *adversarial model* we consider is a persistent passive adversary, able to observe all communication between the client and the server. The adversary aims to recover information about $val(r)$ for the different $r \in R$. In this paper, we consider adversaries that are only exploiting access pattern leakage. Any other leakage that could potentially be exploited is out of the scope of the paper. More specifically, every time the user issues a query, the adversary can detect how many records are included in the response and their ciphertexts.

Definition 3. (Access Pattern Leakage) *If whenever the server responds to a query, the adversary observes the set of all matching identifiers, we say that the scheme allows for access pattern leakage.*

Definition 4. (Communication Complexity) *We consider the communication complexity of a scheme as the number of queries issued by the client and the number of records returned by the server.*

3 Related Work

We have been able to search on encrypted data since the 1980s with *private multiparty computation* [37], and *oblivious random access memory* (*ORAM*) [15]. However, the first paper to consider searching on encrypted data explicitly is by Song, Wagner and Perrig [35] in 2000. Since then, a number of papers have been published on the topic presenting techniques with various leakage profiles, often followed by attacks.

There exist a number of cryptographic techniques that allow searching on encrypted data. These techniques fall broadly in the following categories: *homomorphic encryption* [12, 13, 36], *oblivious random access memory* [15, 17], *private multiparty-computation* [16, 37], *searchable encryption* [5–9, 14, 23], and *property-preserving encryption* [1–3, 31].

Generally, the techniques based on oblivious random access memory and homomorphic encryption give stronger security guarantees, but are fairly slow. Techniques based on property-preserving encryption are fairly efficient, but can have some limitations with regards to security [2]. Searchable encryption lies between the two, being faster than oblivious random access memory and homomorphic encryption based techniques, and allowing for stronger security guarantees than property-preserving encryption based techniques. Using searchable encryption or property preserving encryption, the client can perform a variety of queries on their data: from index-based search queries [8, 9, 23] to more complex ones [10, 11, 32, 33].

In this paper, we focus on access pattern leakage, which is leaked by most systems based on searchable encryption or property preserving encryption [24]. There are a number of attacks that exploit the above leakage. This work started with a seminal paper by Kellaris et al. [24]. They showed that a passive persistent adversary can fully reconstruct a database by observing encrypted range queries and their responses, given a client who performs queries uniformly at random. A few attack papers followed building on this work [20, 28, 29]. There

are also other attacks that focus on slightly different types of leakage. For example, Kornaropoulos et al. [25] utilize leakage from k-nearest neighborhood queries and Grubbs et al. [19] utilize volume leakage. Grubbs et al. [21] also present a snapshot attack.

While most of the above attack papers assume that the client issues queries uniformly at random, in recent work, Kornaropoulos et al. [26, 27] develop distribution-agnostic reconstruction attacks for range and k-nearest neighbor (k-NN) queries. Other attacks in the area assume a more active adversary or that the encryption scheme reveals more properties [4, 22, 34, 38]. There has been some work on mitigating these attacks [10], but it does not prevent asymmetries caused by the client's query distribution. One could use techniques based on ORAM [15, 17] or fully homomorphic encryption [12, 13] to prevent (most of) the above attacks. However, these techniques are quite expensive and not very practical.

4 Blocked Queries

In this section, we present our BQ technique (Figure 1 and Algorithm 1). which aims to introduce an error to any adversary's reconstruction of the database. Without loss of generality, we assume that the range of possible database values, N, is a multiple of a positive integer k, which is a parameter of the scheme. When the client issues query $[a, b]$, the BQ system issues the superset query

$$\left[k \cdot \lfloor a/k \rfloor , k \cdot \lceil (b+1)/k \rceil - 1\right].$$

This way, no two queries can overlap in fewer than k positions, even if $a = b$ in one of them.

Algorithm 1 $BQ(a, b, k)$

1: Return $\left[k \cdot \lfloor a/k \rfloor , k \cdot \lceil \frac{b+1}{k} \rceil - 1\right]$

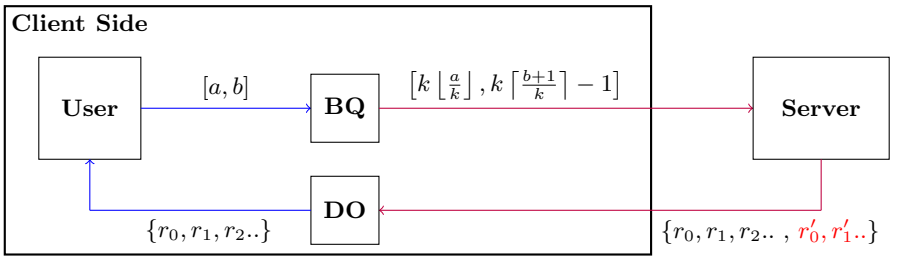


Fig. 1. The BQ technique: When the client wishes to issue a query BQ extends the query to ask for noise values. Once the server responds, the De-Obfuscation module (DO) removes any extra identifiers.

4.1 Analysis

Lemma 1. *Let R_a be the set of records with value in the interval $[a \cdot k, (a+1) \cdot k)$, for some integer a, $0 \leq a < N/k$. If one element of R_a is in a query response, all elements of R_a will also be in the query response.*

Proof. The proof is by contradiction. Let r_1 and r_2 be two records whose values v_1 and v_2 are both in $[a \cdot k, (a+1) \cdot k)$. Suppose there exists some adversary that can deduce that $v_1 < v_2$. The only leakage we consider here is access pattern leakage. In order for the adversary to distinguish between r_1 and r_2, they must observe some query response, say $[c, d]$, which breaks the symmetry. That means that $[c, d]$ returns only one of the two values r_1 or r_2. In order for that to happen an endpoint of the query must fall between v_1 and v_2.

However, c has to be a multiple of k, and d has to be a multiple of k minus 1. Note that $|v_1 - v_2| \leq k - 1$. Additionally, there is only one multiple of k in $[a \cdot k, (a+1) \cdot k)$. Thus, c is either equal to $a \cdot k$ or c is outside $[a \cdot k, (a+1) \cdot k)$ and d is either equal to $(a+1) \cdot k$ or d is outside $[a \cdot k, (a+1) \cdot k)$. We conclude that neither c nor d can fall between v_1 and v_2, and thus, no query can return exactly one of v_1 and v_2. □

Definition 5. (Database D_k) *Let D be a database containing a set of records, (r, x), that the user wishes to store. We construct database D_k by transforming record (r, x) into $\left(r, \left\lfloor \frac{x}{k} \right\rfloor\right)$. See an example in Figure 2.*

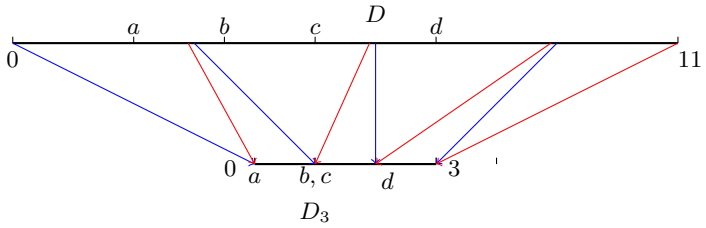


Fig. 2. Example of modified database, D_3, for a database $D = \{a, b, c, d\}$ with values from range $[0, 11]$

Lemma 2. *No adversary A can distinguish between the real database D, and D_k using access pattern leakage.*

Proof. Let A be an adversary that can distinguish between the real database D, and D_k using access pattern leakage. That means that there exists some query response that can be observed when a user queries D that cannot be observed when they query D_k, or there exists some query response that can be observed when a user queries D_k that cannot be observed when they query D.

Let $[a, b]$ be a query whose results can be observed when a user queries D that cannot be observed when they query D_k. If we issue query $[\frac{a}{k}, \frac{b+1}{k} - 1]$ to D_k, we get the same result. Let $[a, b]$ be a query whose results can be observed when a user queries D_k that cannot be observed when they query D. However, query $[a \cdot k, (b+1) \cdot k - 1]$ to D results in the same response. Thus, adversary A

has no way of breaking the symmetry between database D and D_k using access pattern leakage. □

Theorem 1. *Consider a user that issues range queries on an encrypted database D with integer values from interval $[0, \ldots, N-1]$ and adopts the blocked queries mitigation scheme (Algorithm 1). Let k be an integer parameter such that N is multiple of k. Using access pattern leakage, no adversary can identify a range of size smaller than k where any database value belongs in.*

Proof. Suppose there exists some adversary, A, that can identify a range of values smaller than k, where some $val(r)$ belongs. Let $v_1 = val(r)$. Suppose $v_1 \in [a \cdot k, a \cdot (k+1)]$. Suppose that A determines that $v_1 \in [a \cdot k, b)$. Now suppose that there is some record, whose value v_2 is in $[b, a \cdot (k+1)]$. Adversary A could break the symmetry between v_1 and v_2, because A has different information for the two values. Thus, A could break the symmetry between databases D and D_k. This leads to a contradiction by Lemma 2. □

4.2 Communication Overhead

An important point to address is how much the BQ technique increases the communication complexity. When the user wishes to issue query $[a, b]$, we issue instead query $[k \cdot \lfloor a/k \rfloor, k \cdot \lceil (b+1)/k \rceil - 1]$. The user asked us to query $b - a$ values in the database, and at most, we will query $b - a + 2k$ values in the database.

Let $\bar{s}$ be the average query size that the client issues. Using our scheme, the client can introduce an error of k to the adversary's reconstruction, while the fraction of the size of the noise to normal traffic is bounded by $\frac{2k}{\bar{s}}$. The overhead of this scheme depends a lot on the choice of k, and the query distribution that the user picks. For example, if the user picks queries uniformly at random, the average query size is

$$\frac{2}{N(N+1)} \sum_{i=1}^{N} i(N-i+1) = \frac{2}{N(N+1)} \frac{(N+2)N(N+1)}{6} = \frac{N+2}{3}.$$

Thus, the fraction of noisy records to normal ones is $\frac{6k}{N+2}$. If $\frac{N}{k}$ is a constant, then the multiplicative communication overhead of this scheme is also a constant, while the adversary's reconstruction error is proportional to N.

The trivial mitigation strategy would be to ask for the whole database back on every query. This is equivalent to setting $k = N$. Our scheme allows the user to reduce the communication overhead by making a trade-off on security. Generally, as long as $k << N/6$, the scheme is more efficient over the trivial scheme.

5 Wrap-Around Queries

An assumption commonly made in the literature on attacks on encrypted databases is that the adversary knows the query distribution. In particular, for 1D range

queries, the distribution is often assumed to be "uniform at random," i.e., all queries $[a, b], a \leq b$, have the same probability of being issued. To defend against these adversaries, in this section, we present our second mitigation scheme, *wrap-around queries*, schematically illustrated in Figure 3.

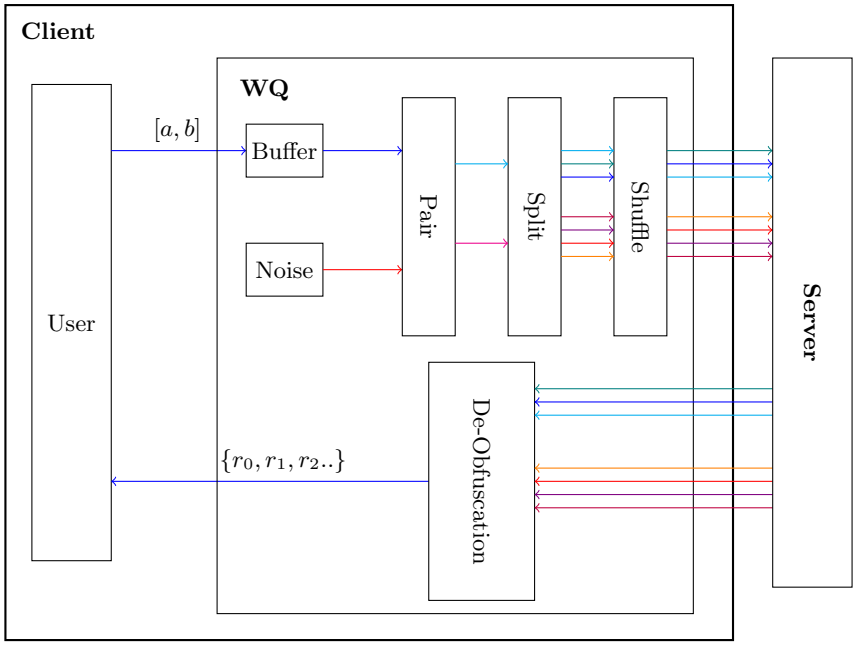


Fig. 3. The wrap-around queries module receives queries from the client and adds them to a buffer. It periodically constructs a pair of queries that contains both noise (i.e., fictitious queries) and real queries. Each query in the pair is further split into singleton queries (i.e., queries spanning a single value). Then, the singleton queries from each range query are shuffled. The shuffled queries are sent to the server. The de-obfuscation component receives the answers to the queries in the pair from the server, filters out the answers to noise, and reassembles the remaining singleton answers into answers to the original queries, which it forwards to the client.

This scheme turns the linear range of possible database values into a cyclic range with no discernible beginning and end in the eyes of the adversary. As the user issues queries according to a fixed probability distribution, we add fictitious queries that correspond to a "complementary" distribution. Thus, we convert the original query stream into one uniformly distributed over the cyclic range, hence removing any asymmetries on the number of times database values are present in query answers. As a result, we prevent the adversary from distinguishing among any two cyclic shifts of the database using the access pattern leakage.

5.1 Key Ideas

The wrap-around queries technique depends on three key ideas:

1. Our technique issues additional queries to introduce noise. Approximately, every time the client issues a query, we issue one or more additional fictitious queries. The fictitious queries may be normal queries (i.e., $[a', b']$ where $a' \leq b'$) or wrap-around queries (i.e., $[a', b']$ where $a' > b'$).
2. Our technique issues queries in pairs. To determine if a slot in the pair contains a noise or a normal query, we flip a coin.
3. Whenever a query $[a, b]$ is to be issued, we issue a series of *singleton queries* $[a, a], [a + 1, a + 1] \ldots [b, b]$, instead.

This technique essentially changes the ordered vector of values $[1, N]$ into a circular buffer of N values that has no start and no end. This change (from a vector to a circular buffer) renders a persistent passive adversary unable to determine which identifier corresponds to the minimum (or maximum) value. In addition, the fictitious queries follow such a distribution so that all individual values $val(r_i)$ have the same frequency of appearance.

5.2 Example

We illustrate how the wrap-around queries scheme works with an example. Suppose we have a database of size 5 and the user issues queries that are uniformly distributed. Let S_n be the set of normal queries that the client can issue (i.e. $[a, b]$ where $a \leq b$) and let S_w be the set of wrap-around queries that the client cannot issue (i.e. $[a, b]$ where $a > b$). The set S_n contains 15 queries and the set S_w contains 10 queries. All these queries are shown in Table 1.

(a) Normal					(b) Wrap-around				
$[1,1]$	$[2,2]$	$[3,3]$	$[4,4]$	$[5,5]$	-	-	-	-	-
$[1,2]$	$[2,3]$	$[3,4]$	$[4,5]$	-	-	-	-	-	$[5,1]$
$[1,3]$	$[2,4]$	$[3,5]$	-	-	-	-	-	$[4,1]$	$[5,2]$
$[1,4]$	$[2,5]$	-	-	-	-	-	$[3,1]$	$[4,2]$	$[5,3]$
$[1,5]$	-	-	-	-	-	$[2,1]$	$[3,2]$	$[4,3]$	$[5,4]$

Table 1. All possible cyclic queries on five elements: (a) normal queries; (b) wrap-around queries

We use a buffer, B, to store the queries that the user wishes to issue. Let's acquire a biased coin that returns heads with probability $\frac{15}{25}$ and tails with probability $\frac{10}{25}$. We flip the biased coin twice. On heads, we pick a range query from buffer B (if the buffer is empty we choose a normal query from S_n uniformly at random) and on tails we pick a fictitious query, which is a wrap-around query selected uniformly at random from S_w. Thus, after each coin flip, we add a query to the pair. Once the pair is formed, we deconstruct all queries into a series of singleton queries, shuffle the singletons, and issue them to the server. Once the server responds, we identify the responses corresponding to real queries and send them to the user.

In Table 2, we show how many times each single value (from 1 to 5) is expected to be included in a query in the pair. The probability of value 1 appearing in a query in the pair is $15/25 \cdot 5/15 + 10/25 \cdot 10/10 = 3/5$. The probability of value

3 appearing in a query in a given slot in the pair is $15/25 \cdot 9/15 + 10/25 \cdot 6/10 = 3/5$: exactly the same as the probability of value 1. The same happens for all values: the probability of a value i to appear in a query in a given slot is $3/5$: the same probability for all values i.[4] Thus, all values have the same probability of being included in a query in a slot.

Value	Appearances	Normal Queries
1	5	[1,1], [1,2], [1,3], [1,4], [1,5]
2	8	[1,2], [1,3], [1,4], [1,5], [2,2], [2,3], [2,4], [2,5]
3	9	[1,3], [1,4] [1,5], [2,3], [2,4], [2,5], [3,3], [3,4], [3,5]
4	8	[1,4], [1,5], [2,4], [2,5], [3,4], [3,5], [4,4], [4,5]
5	5	[1,5], [2,5], [3,5], [4,5], [5,5]

Value	Appearances	Wrap-around Queries
1	10	[5, 1], [4, 1], [3, 1], [2, 1], [5, 2], [4, 2], [3, 2], [5, 3], [4, 3], [5, 4]
2	7	[5, 2], [4, 2], [3, 2], [5, 3], [4, 3], [5, 4], [2, 1]
3	6	[2, 1], [3, 1], [3, 2], [4, 3], [5, 3], [5, 4]
4	7	[2, 1], [3, 1], [4, 1], [3, 2], [4, 2], [4, 3], [5, 4]
5	10	[2, 1], [3, 1], [4, 1], [5, 1], [3, 2], [4, 2], [5, 2], [4, 3], [5, 3], [5, 4]

Table 2. All values in the database appear with the same frequency in the query results. In this table we show in how many queries each value appears. That is, value "1" appears in 5 normal queries and 10 wrap-around queries (a total of 15). Similarly, value "2" appears 8 times in normal queries and 7 times in wrap-around queries (a total of 15). Each values appears 15 times in the normal and wrap-around queries combined.

5.3 Algorithm

In this section, we present Algorithm 2, WQ, which performs a generic version of our wrap-around mitigation scheme. The algorithm takes the following inputs:

- *buffer*: a buffer that contains the queries that the client wishes to issue;
- *coin*: a random coin with given fixed probabilities for heads and tails;
- S_n: set of normal queries;
- S_w: set of wrap-around queries;
- N: size of the database.

Note that inputs *coin*, S_n, and S_w depend on the probability distribution of the user's queries.

Algorithm WQ operates as follows. As long as there are still queries in *buffer*, WQ generates a new pair of queries. For each slot in the pair, we flip *coin*, and depending on the result, either a normal or a wrap-around query takes up the slot. If *coin* instructs that a normal query takes up the slot, we pop one from *buffer*. If *buffer* is empty, we just pick a normal query uniformly at random. If *coin* instructs that a wrap-around query takes up the slot, we pick

[4] The reader might wonder that since there are five values (1 to 5), then each value should have probability 1/5 (not 3/5) to appear in the query results. We should note however that these queries are *range* queries that return more than one value.

one uniformly at random. Each query is then split into singleton queries, and a random permutation of the singletons is added to the pair slot.

Once the pair is full, we query the server. We discard any answers to wrap-around or fake queries, and return the rest to the client.

Algorithm 2 $WQ(buffer, coin, S_n, S_w, N)$

```
1: while |buffer| > 0 do
2:    pair = []
3:    for  two rounds do
4:       Flip coin
5:       if coin = 1 then
6:          [a, b] = buffer.pop()
7:          // If buffer is empty, pick query [a, b] uniformly at random from S_n
8:       else
9:          Pick query [a, b] uniformly at random from S_w
10:
11:      singletons = []
12:      for i ∈ range(a, b) do
13:         Add query [i, i] to singletons
14:      Add a random permutation of singletons to pair
15:
16:   Issue all queries from pair to server
17:
18:   if pair contained a user query then
19:      Return the relevant server's responses to the user.
```

5.4 Uniform Query Distribution

For the case of a user that issues uniformly distributed range queries, Algorithm 2 is specialized by selecting a biased coin, c, with probability

$$Pr(c = 1) = \frac{N(N+1)}{2N^2} = 1 - Pr(c = 0),$$

which yields Algorithm 3, WQU.

Algorithm 3 $WQU(buffer, N)$

```
1: Let c be a biased coin with Pr(c = 1) = N(N+1)/(2N^2) = 1 − Pr(c = 0)
2: Let S_n be the set of queries [i, j] such that 1 ≤ i ≤ j ≤ N
3: Let S_w be the set of queries [i, j] such that 1 ≤ j < i ≤ N
4: Execute WQ(buffer, c, S_n, S_w, N) (Algorithm 2)
```

We now analyze the security properties of Algorithm 3.

Lemma 3. *When using Algorithm 3, a slot of the pair contains each query with the same probability.*

Proof. Table 3 shows all possible queries.

Table 3. All possible queries: Note that the blue queries are normal, and the red ones are wrap-around.

$[1,1]$	$[1,2]$	...	$[1,N-1]$	$[1,N]$
$[2,1]$	$[2,2]$	...	$[2,N-1]$	$[2,N]$
$[3,1]$	$[3,2]$	...	$[3,N-1]$	$[3,N]$
...	...	...	...	...
$[N-1,1]$	$[N-1,2]$	...	$[N-1,N-1]$	$[N-1,N]$
$[N,1]$	$[N,2]$	...	$[N,N-1]$	$[N,N]$

Let q_1 be a normal query. Note that there are $\frac{N(N+1)}{2}$ normal queries. In a given pair slot, query q_1 is issued with probability

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

Now, let q_2 be a wrap-around query. Note that there are $N^2 - \frac{N(N+1)}{2}$ wrap-around queries. In a given pair slot, query q_2 is issued with probability

$$\frac{N^2 - \frac{N(N+1)}{2}}{N^2} \cdot \left(\frac{1}{N^2 - \frac{N(N+1)}{2}}\right) = \frac{N(N-1)}{2N^2} \cdot \left(\frac{2}{N(N-1)}\right) = \frac{1}{N^2}$$

Thus, all queries are issued with the same probability $\frac{1}{N^2}$. □

Lemma 4. *When using Algorithm 3, each value in the database has the same probability of being in a given pair slot.*

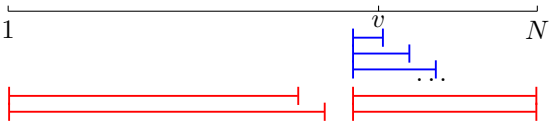
Proof. Lemma 3 shows that all queries are issued with the same probability.

Let's look at all the queries that query some value, say v. There are N queries that start at v and contain v.

$$[v,v],[v,v+1],[v,v+2]\ldots[v,v-3],[v,v-2],[v,v-1]$$

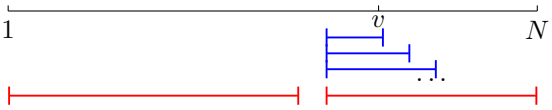
There are $N-1$ queries that start at $v-1$ and contain v:

$$[v-1,v],[v-1,v+1],[v-1,v+2]\ldots[v-1,v-3],[v-1,v-2]$$



There are $N-2$ queries that start at $v-2$ and contain v:

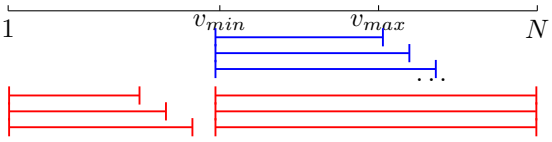
$$[v-2,v],[v-2,v+1],[v-2,v+2]\ldots[v-2,v-3]$$



Thus, there are $N + N - 1 + N - 2 + \ldots + 1 = \frac{N(N+1)}{2}$ queries that query value v. So, for any v there are $\frac{N(N+1)}{2}$ queries that query it. Since all queries are issued with the same probability (Lemma 3), all values v have the same probability of being queried in given a slot pair. □

Lemma 5. *When using Algorithm 3, every contiguous set of points of size s has the same probability of being queried in a given pair slot.*

Proof. Let's look at some set of records S, and at the queries that return all elements of the set. Suppose that the smallest value of an element in S is v_{min} and the largest is v_{max}. There are $N - s$ queries that return all elements of S and start at v_{min}.



$$[v_{min}, v_{max}], [v_{min}, v_{max} + 1], [v_{min}, v_{max} + 2] \ldots$$
$$[v_{min}, v_{min} - 3], [v_{min}, v_{min} - 2], [v_{min}, v_{min} - 1].$$

Similarly to Lemma 4, there are $\frac{(N-s)(N-s+1)}{2}$ queries that return all points from S. Thus, for any set of size s there are $\frac{(N-s)(N-s+1)}{2}$ queries that return all its elements. □

Definition 6. (Database D_{shift_s}) *Let D be a database with integer values from an interval of size N and let s be an integer. We define database D_{shift_s} as the set of records $(r, x + s(mod\ N))$, for all records $(r, x) \in D$. That is, we cyclically shift the values of all records by s.*

The following theorem summarizes the security property of wrap-around queries method for a uniform query distribution.

Theorem 2. *Consider a user that issues uniformly distributed range queries on an encrypted database D with integer values from an interval of size N and adopts the wrap-around mitigation scheme (Algorithm 3). For any integer s, a passive persistent adversary who observes the noisy query stream produced by the algorithm and the corresponding query results cannot distinguish with probability greater than $1/2$ between database D and database D_{shift_s} using access pattern leakage and knowledge of the user's query distribution.*

Proof. Suppose there exists some adversary that can distinguish between D and D_{shift_s} with probability greater than $1/2$. The adversary has two weapons: (i) the

fact that a user that issues uniformly distributed range queries, and (*ii*) access pattern leakage. As long as there are queries that the user wants to send out, Algorithm 3 will create pairs of queries to send to the server.

Algorithm 3 issues range queries as a random permutation of singleton queries. Note that the random permutation is necessary, as otherwise we leak the order of the records and the distances between them on a single query. We have to send the queries in pairs. If we sent only one query at a time, the adversary would know that the last query we sent was a normal query.

1. Let us first examine the access pattern leakage: Algorithm 3 will eventually issue all queries $[a, b]$ for $a, b \in [1, N]$ regardless of whether the database stored is D or D_{shift_s}. Now, because Algorithm 3 issues both normal and wrap-around queries, we can map a query from D to a query from D_{shift_s} with the same response. Query $[a, b]$ on database D produces the same response as query $[a+s(\text{mod } N), b+s(\text{mod } N)]$ on database D_{shift_s}. Thus, the adversary cannot get any information out of the access pattern leakage as the two databases produce the same sets of responses.
2. Now, let us examine what the adversary can accomplish using the knowledge that the user issues uniformly distributed range queries. The adversary might try to use this knowledge since due to the uniformity of queries, the frequency with which values of the database appear in query answers varies. To be exact, elements in the middle of the database have a higher chance of being queried than elements in the ends. However, Algorithm 3, ensures that any set of contiguous points of a certain size is equally likely to be in a pair slot (Lemma 5). Thus, there are no asymmetries that the adversary could exploit to deduce which database is which.

Thus, neither uniformity nor access pattern leakage can help the adversary break the symmetry between the two databases, and the best they can do is guess, which succeeds with probability $1/2$. □

Corollary 1. *Consider a user that issues uniformly distributed range queries on an encrypted database D with integer values from an interval of size N and adopts the wrap-around mitigation scheme (Algorithm 3). The adversary is unable to infer which record has the minimum (or maximum) value with probability greater than $1/n$, where n is the size of the database.*

Notably, after all their observations the adversary gains no knowledge on which record is the first one.

5.5 Truncated Uniform Distribution

The wrap-around queries mitigation technique can be extended to work with any fixed query distribution. Indeed, from any fixed distribution, we can construct a complementary distribution such that an adversary who observes the resulting noisy query stream sees the equivalent of a uniform distribution over the queries for a cyclic buffer (Table 3).

In this section, we consider a client who issues queries of up to size t uniformly at random, i.e., a truncated uniform distribution. This is in line with related work [26, 27] that also considers a client who focuses on queries of small size.

In this case, we can instruct our mitigation scheme to also only send queries of size up to t. That is, the only queries allowed (for both the client and the algorithm) are of the form $[a, b]$, where: if the query is normal ($a \leq b$): $b - a$ has to be smaller or equal to t, else if the query is a wrap-around query: $b + N - a$ has to be smaller or equal to t.

The new algorithm, Algorithm 4, WQT, has to use a new biased coin, and modified S_n and S_w sets. The new biased coin will return 1 with probability $\frac{2N-t+1}{2N}$. We set S_n to the set of normal queries $[a, b]$, where $b - a \leq t$, and S_w to the set of wrap-around queries $[a, b]$, where $b + N - a \leq t$.

Algorithm 4 $WQT(buffer, N, t)$

1: Let c be a biased coin with $Pr(c = 1) = \frac{2N-t+1}{2N} = 1 - Pr(c = 0)$
2: Let S_n be the set of queries $[i, j]$ such that $1 \leq i \leq j \leq N$ and $j - i \leq t$
3: Let S_w be the set of queries $[i, j]$ such that $1 \leq j < i \leq N$ and $j + N - i \leq t$
4: Execute $WQ(buffer, c, S_n, S_w, N)$ (Algorithm 2)

We now analyze the security properties of Algorithm 4.

Lemma 6. *When using Algorithm 4, a slot of the pair contains each query with the same probability.*

Proof. There are $\sum_{i=0}^{t-1} N - i = \frac{1}{2}t(2N - t + 1)$ normal queries, and $\sum_{i=0}^{t-1} i = \frac{1}{2}t(t - 1)$wrap-around queries. In a given pair slot, a normal query and a wrap-around query are issued with probability

$$\frac{2N - t + 1}{2N} \cdot \frac{2}{t(2N - t + 1)} = \frac{1}{Nt} \text{ and } \left(1 - \frac{2N - t + 1}{2N}\right) \cdot \frac{2}{t(t - 1)} = \frac{1}{tN},$$

respectively. Thus, all queries are issued with the same probability. □

Lemma 7. *When using Algorithm 4, every contiguous set of points of size s has the same probability of being queried in a given pair slot.*

Proof. For any set of size $s \leq t$, the proof follows as in Lemma 5 supported by Lemma 6, there are $\frac{(N-s)(N-s+1)}{2}$ queries that return all its elements. For any set of size $s > t$ there are 0 queries that return all its elements.

Theorem 3. *Consider a user that issues uniformly distributed range queries of size smaller than t, on an encrypted database D with integer values from an interval of size N and adopts the wrap-around mitigation scheme (Algorithm 4). For any integer s, a passive persistent adversary who observes the noisy query stream produced by the algorithm and the corresponding query results cannot distinguish with probability greater than $1/2$ between database D and database D_{shift_s} using access pattern leakage and knowledge of the user's query distribution.*

The proof follows as in Theorem 2 supported by Lemma 7.

5.6 Communication Overhead

In order to analyze the communication overhead introduced by the wrap-around queries mitigation method, we consider a client who performs queries up to size t uniformly at random and employs Algorithm 4, WQT. Note that for the special case $t = N$, we have a uniform query distribution.

The communication overhead depends heavily on the underlying database. The protocol is most efficient on *dense* databases. Here, we have overhead from singleton queries and from wrap-around queries as before. In the setting of a dense database, there is at least one record for every value queried. Thus, the multiplicative overhead introduced is at most $2\times$.

Let us now consider the overhead introduced from wrap-around queries.The average size of a normal query is $\frac{\sum_{i=1}^{t} i(N-i+1)}{\sum_{i=1}^{t} N-i+1} = \frac{(t+1)(3N-2t+2)}{3(2N-t+1)}$, and the average size of a wrap-around query is $\frac{2\sum_{i=0}^{t-1} i(i+1)}{t(t-1)} = \frac{2(t+1)}{3}$.

Thus, the issued queries' average size is:

$$\frac{2N-t+1}{2N}\frac{(t+1)(3N-2t+2)}{3(2N-t+1)} + \frac{t-1}{2N}\frac{2(t+1)}{3} = \frac{t+1}{2}.$$

Now, the multiplicative overhead of the query, denoted with m, is the ratio between the new average query size and the average size of normal queries:

$$m = \frac{\frac{t+1}{2}}{\frac{(t+1)(3N-2t+2)}{3(2N-t+1)}} = \frac{3}{2} \cdot \frac{2N-t+1}{3N-2t+2}.$$

Note that depending on the choice of t. we have $1 \leq m < 1.5$, hence the smaller t, the smaller the overhead. Thus, the communication overhead varies between $1\times$ and $3\times$, depending on t and how many records are on each value. The more records per value, the smaller the proportional overhead.

5.7 Defending Against Current Attacks

The WQ technique makes the adversary's job more difficult in two ways:

1. It removes asymmetries due to the client's query distribution: Exploiting these asymmetries has been a focus of Kellaris et al. [24] in their seminal paper on attacks. Grubbs et al. [20] present a very elegant algorithm as well that exploits this uniformity assumption. As we remove the asymmetries, these attacks can no longer reconstruct the database.
2. More importantly, it turns the database into a cyclic buffer in the eyes of any adversary: No adversary exploiting access pattern leakage can tell which element is the first (or last) in the database. So, even if an adversary can reconstruct the shape of the database, they won't be able to recover the actual record values, unless they have access to auxiliary information.

5.8 Observations

Encryption schemes that allow for range queries can be costly. It's interesting to note that when running Algorithm 2, the server is never queried for a range that is larger than 1. Thus, a lot of the formerly required machinery is no longer necessary, the server can just store an encrypted dictionary.

Regarding storage complexity, Algorithm 2 issues singleton queries from two range queries. This can require a lot of storage. To reduce this complexity, we can run Algorithm 2 in a streaming fashion. Specifically, instead of constructing and storing *pair*, we can use techniques for block ciphers with arbitrary block size to perform the pseudorandom permutation in sub-linear (in N) space, see for example [18, 30].

6 Conclusion

In recent years, a number of attacks have been developed to fully or approximately reconstruct encrypted databases from the information leakage of encrypted range queries and their responses. In an effort to mitigate the attacks, this paper presents two approaches to help better protect the confidentiality of user data. Our first approach, blocked queries, introduces an error to the reconstruction of the database by any adversary. Our second approach, wrap-around queries, is aimed specifically at a broad class of attacks that assume the user queries the database uniformly at random. This approach removes exploitable asymmetries in query answers caused by uniformity, thus reducing the capability of such attacks.

Acknowledgments

We are grateful to Arkady Yerukhimovich for valuable comments and suggestions.

References

[1] Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proc. ACM Int. Conf. on Management of Data, SIGMOD (2004)
[2] Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Advances in Cryptology, CRYPTO (2007)
[3] Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Advances in Cryptology, EUROCRYPT (2009)
[4] 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)

[5] Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Advances in Cryptology, CRYPTO (2013)
[6] Chang, Y.C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Applied Cryptography and Network Security, ACNS (2005)
[7] Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Advances in Cryptology, pp. 577–594, ASIACRYPT (2010)
[8] Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. In: Proc. ACM Conf. on Computer and Communications Security, CCS (2006)
[9] Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security **19**(5), 895–934 (2011)
[10] 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)
[11] Faber, S., Jarecki, S., Krawczyk, H., Nguyen, Q., Rosu, M., Steiner, M.: Rich queries on encrypted data: Beyond exact matches. In: Proc. European Symp. on Research in Computer Security, ESORICS (2015)
[12] Gentry, C.: Computing arbitrary functions of encrypted data. Commun. ACM **53** (2010)
[13] Gentry, C., Boneh, D.: A fully homomorphic encryption scheme, vol. 20:09. Stanford university Stanford (2009)
[14] Goh, E.J.: Secure indexes. Cryptology ePrint Archive, Report 2003/216 (2003), https://eprint.iacr.org/2003/216
[15] Goldreich, O.: Towards a theory of software protection and simulation by oblivious RAMs. In: Proc. ACM Symp. on Theory of Computing, STOC (1987)
[16] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proc. ACM Symp. on Theory of Computing, STOC (1987)
[17] Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. Journal of the ACM (JACM) **43**(3), 431–473 (1996)
[18] Granboulan, L., Pornin, T.: Perfect block ciphers with small blocks. In: Int. Workshop on Fast Software Encryption, FSE (2007)
[19] Grubbs, P., Lacharite, 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)
[20] 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
[21] Grubbs, P., Ristenpart, T., Shmatikov, V.: Why your encrypted database is not secure. In: Proc. Workshop on Hot Topics in Operating Systems, HotOS (2017)

[22] Grubbs, P., Sekniqi, K., Bindschaedler, V., Naveed, M., Ristenpart, T.: Leakage-abuse attacks against order-revealing encryption. In: Proc. IEEE Symp. on Security and Privacy, SP (2017)
[23] Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proc. 2012 ACM Conf. on Computer and Communications Security, CCS (2012)
[24] Kellaris, G., Kollios, G., Nissim, K., O'Neill, A.: Generic attacks on secure outsourced databases. In: Proc. ACM Conf. on Computer and Communications Security, CCS (2016)
[25] 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, SP (2019)
[26] 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
[27] 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
[28] 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)
[29] Markatou, E.A., Tamassia, R.: Full database reconstruction with access and search pattern leakage. Cryptology ePrint Archive, Report 2019/395 (2019), https://eprint.iacr.org/2019/395
[30] Morris, B., Rogaway, P., Stegers, T.: Deterministic encryption with the Thorp shuffle. J. Cryptology **31**(2), 521–536 (2018)
[31] Pandey, O., Rouselakis, Y.: Property preserving symmetric encryption. In: Advances in Cryptology, EUROCRYPT (2012)
[32] Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S.G., George, W., Keromytis, A., Bellovin, S.: Blind Seer: A scalable private DBMS. In: Proc. IEEE Symp. on Security and Privacy, SP (2014)
[33] Poddar, R., Boelter, T., Popa, R.A.: Arx: A strongly encrypted database system. IACR Cryptology ePrint Archive **2016**, 591 (2016)
[34] 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)
[35] Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proc. IEEE symp. on Security and Privacy, SP (2000)
[36] Vaikuntanathan, V.: Computing blindfolded: New developments in fully homomorphic encryption. In: Proc. IEEE Symp. on Foundations of Computer Science, FOCS (2011)
[37] Yao, A.C.: Protocols for secure computations. In: Proc. IEEE Symp. on Foundations of Computer Science, FOCS (1982)
[38] 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)