# Low-Cost Hiding of the Query Pattern

Maryam Sepehri
University of Waterloo
Waterloo, Canada
maryam.sepehri@uwaterloo.ca

Florian Kerschbaum
University of Waterloo
Waterloo, Canada
florian.kerschbaum@uwaterloo.ca

## ABSTRACT

Several attacks have shown that the leakage from the access pattern in searchable encryption is dangerous. Attacks exploiting leakage from the query pattern are as dangerous, but less explored. While there are known, lightweight countermeasures to hide information in the access pattern by padding the ciphertexts, the same is not true for the query pattern. Oblivious RAM hides the query patterns, but requires a logarithmic overhead in the size of the database and hence will become even slower as data grows. In this paper we present a query smoothing algorithm to hide the frequency information in the query pattern of searchable encryption schemes by introducing fake queries. Our method only introduces a constant overhead of 7 to 13 fake queries per real query in our experiments. Furthermore, we show that our query smoothing algorithm can also be applied to range-searchable encryption schemes and then prevents all recent plaintext recovery attacks.

## CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

## KEYWORDS

Queries over Encrypted Data; Leakage; Range Queries

**ACM Reference Format:**
Maryam Sepehri and Florian Kerschbaum. 2021. Low-Cost Hiding of the Query Pattern. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Virtual Event, Hong Kong*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3433210.3453103

## 1 INTRODUCTION

When outsourcing databases to the cloud, users may want to apply another layer of protection by encryption. Searchable encryption [7, 9, 19, 20] allows efficient search over encrypted data where each data item is probabilistically encrypted. Hence, different from property-preserving encryption [18, 40] a snapshot attacker cannot obtain any information useful for recovering plaintexts. However, the access and query (also called search) pattern may leak additional information about the data to a persistent adversary that controls the database system in the cloud.

Hiding the access pattern (i.e. information about the search result) incurs a high cost making most database systems impractical. For example, Oblivious RAM [12, 44] can hide the access and query pattern (i.e. information about which queries repeat), but incurs a logarithmic overhead in the database size, i.e. the larger the database, the slower each ORAM query. The logarithmic overhead of online ORAM has been proven as a lower bound in the information-theoretic [12], cryptographic [31] and differentially private setting [38].

Yet, completely hiding the access pattern may not be necessary. Instead, we can aim for a weaker, but more efficient protection mechanism. There are known, low-cost techniques for hiding the frequency and volume information in access patterns [22, 23, 25]. These pad the ciphertexts or query results by dummy values smoothing the frequency and even co-occurrence frequency distribution of keywords. However, by themselves these techniques are insufficient, since the query pattern may still leak critical information. If the query is deterministically encrypted and the adversary knows the distribution of queries, one can infer the queried keywords and hence recover the plaintext of the database. Even if the query is probabilistically encrypted, the access pattern will reveal the query distribution in a somewhat static database, since the access pattern is overlapping for identical query keywords, but non-overlapping for distinct query keywords. This leakage from the query pattern has been used in recent attacks on range queries [13, 14, 17, 24, 28] that are very efficient in recovering the approximate plaintext using a number of queries only linear in the database size.

In this paper we present the analysis of a low-cost query smoothing algorithm that hides the frequency information in the query pattern. We intersperse fake queries with real queries in order to hide the query pattern – a technique often casually proposed as a countermeasure without formal evaluation. We define a security notion we call $d$-smooth query pattern where at least $d$ queries have the same frequency. While this notion is weaker than the security achieved by ORAM our analysis shows that we only need a constant number of fake queries per real query independent of the database size. Therefore, differently than ORAM, our method can scale to very large data sets.

Furthermore, since when using our query smoothing algorithm in combination with the above techniques neither the access nor the query pattern leaks frequency information about the database, we can thwart all recent attacks on encrypted range queries [13, 14, 17, 24, 28]. By interspersing cleverly chosen, fake queries we can void a crucial assumption of these attacks rendering them entirely unsuccessful. We describe our results in Section 6.


*ASIA CCS '21, June 7–11, 2021, Virtual Event, Hong Kong*

ACM ISBN 978-1-4503-8287-8/21/06…$15.00
https://doi.org/10.1145/3433210.3453103

We implemented our method and a query takes only 1.7 ms on a database of size $2^{16}$ keywords whereas ORAM already takes 47 ms on the same database. In summary our contributions are follows

- We define the notion of $d$-smooth query pattern.
- We present a query smoothing algorithm that achieves a $d$-smooth query pattern.
- In our theoretical analysis we show that our algorithm incurs a constant $O(1)$ overhead per query independent of the database size.
- In our experimental evaluation we show that our algorithm withstands a frequency analysis attack and its practical overhead is small compared to ORAM.
- We apply our technique to secure range queries and show that it is capable of preventing all known attacks.

The remainder of the paper is structured as follows. We describe related work, in particular leakage-abuse attacks, in Section 2. Then we describe how the query pattern leaks information and can be used to perform attacks in Section 3. In Section 4 we present our query smoothing algorithm and its theoretical evaluation. We summarize our practical evaluation including security and performance analysis compared to ORAM in Section 5. In Section 6 we apply our query smoothing algorithm to attacks on range queries before we present our conclusions of the research in Section 7.

## 2 RELATED WORK

Data protection by encryption raises the problem of developing efficient techniques for supporting queries over encrypted data. A number of different base technologies for encrypted query processing have been proposed: homomorphic encryption, oblivious RAMs (ORAM) [12, 44], secure multi-party encryption, searchable and structured encryption, deterministic encryption and order-preserving encryption (OPE). These techniques can be broadly classified into techniques that have leakage to a snapshot adversary (deterministic and order-preserving encryption), only to a persistent adversary (searchable encryption) and no leakage. Systems based on secure search algorithms on fully homomorphic encryption [3] with full security and secure multi-party computation with no leakage [4], still incur a significant overhead rendering them mostly impractical. Systems with leakage to a snapshot adversary [18, 40] are broken very easily [16, 35]. Also, systems with leakage only to a persistent adversary can be broken [5, 22] which has again been shown to be easy in case of range queries [13, 14, 17, 24, 28]. Particularly, the attacks on range queries put into question whether efficient search over encrypted data, i.e. with a little leakage, can be securely implemented. The most effective attack for plaintext recovery is frequency analysis [29] and techniques to hide the frequency leaked from the access pattern [22, 23, 25] and the ciphertexts [26, 27, 30] have been developed. However, despite numerous attempts at creating secure and efficient queries over encrypted data none of the proposed methods is secure against leakage from the query pattern which we address in this paper.

De Capitani di Vimercati et al. [8] introduce an access hiding technique based on a shuffle index structure, which adopts a distributed $B^+$ tree. Their scheme hides user queries by fake ones, a cache and shuffles the content among blocks stored at the server. However, their security and overhead analysis does not show concrete trade-offs that compare to our work or ORAM, e.g. they uniformly choose a user-defined number of non-overlapping fake queries which without shuffling does not prevent frequency leakage from the queries. Whereas we provably limit the disclosure of frequency information using only fake queries and a cache.

We briefly review the most important attacks on searchable encryption. The first inference attack on searchable encryption [22] successfully exploits access pattern leakage and uses prior knowledge about the frequency of keyword pairs and search queries to recover all of the plaintext of the encrypted database. Later Cash et al. [5] improved the accuracy of this plaintext recovery attack based on additional knowledge of the query distribution when the search space becomes large. Liu et al. [32] investigated the leakage of searchable encryption schemes from the query pattern that helps an adversary to distinguish two queries generated from the same keyword and showed that it can be used to recover the plaintext of all queries and hence the database. They propose a countermeasure called grouping-based construction to make the query result as uniform as possible by querying all keywords in a group once a keyword is queried whereas we query each keyword in a group the same number of times. Furthermore, their approach is ad-hoc i.e. it comes without a formal security guarantee. The first query identification attack proposed by Oya et al. [36] that exploits access and search pattern leakage, as well as some background information and query distribution to uncover the underlying keywords of user queries. Their attack follows a maximum likelihood estimations approach to find the most likely keyword of each received query. Further active attacks on searchable encryption have been proposed [15, 47], but are out of scope of this paper.

Recent works [13, 14, 17, 24, 28] presented generic attacks of data recovery against encryption schemes that support range queries, including ORAM. As demonstrated by Kellaris et al. [24] full data recovery is possible in the case where the distribution of range queries is known. This generic attack results in a complete reconstruction of the exact values of every record in a database. The authors use two sources of leakage: access pattern and communication volume. We describe this attack in detail and show how to counter it in Section 6. Note that our scheme still leaks both the access pattern and the result volume. However, this leakage is uncritical and useless for the attacks when combined with our techniques using fake queries, since order reconstruction becomes infeasible.

More recently, Lacharite et al. [28] improved the lower bounds for the number of queries required for the full reconstruction of database values. Using the same setting for full reconstruction as [24], the authors showed that for dense databases the expected number of queries is only $N \log N + O(N)$, where $N$ is the number of distinct plaintext values. They devised three attacks: (i) full reconstruction with rank information and using only access pattern leakage similar to the one proposed in [24], (ii) approximate reconstruction attack for recovering the plaintext of every record up to a marginal error with only $O(N)$ queries, and (iii) reconstruction attack to recover the plaintext without making any assumption about the distribution of those values that is practical in many real-life applications. Grubbs et al. [14] show that plaintext recovery attacks are feasible on range queries with only precise volume leakage,

i.e. no query or access pattern leakage. Note that ORAM also leaks this precise volume information of results sets and is hence susceptible to this type of attack. Grubbs et al. [13] generalize the generic attack from [24] and show that using algorithms from PAC learning even more efficient approximate attacks are feasible. Gui et al. [17] show attacks that still work when query ranges have limited size. All of these attacks [13, 14, 17, 28] rely on the same assumption and the same reconstructed order leakage as the attack by Kellaris et al. [24] for which we provide a countermeasure, i.e. preventing order reconstruction, and hence our countermeasure presented in Section 6 is also successful against those including those using only volume leakage.

## 3 PROBLEM STATEMENT

We consider a database with one client $C$ and one server $S$, possibly remotely placed into the cloud. This setup can be generalized to multiple clients, but this is out of scope of this paper. The server $S$ maintains a database $D$ which is searchable over a single attribute $A$. We begin with keyword (point) queries and later extend that to range queries in Section 6. The attribute $A$ is populated with keyword values $a_1, \ldots, a_m$. We assume that frequency smoothing has already been applied to the attribute, that is the underlying keyword values distribution is uniform, and that each keyword is returning a result of fixed size, i.e. the size of the database is $O(\text{poly}(m))$. Better techniques exist in the literature [22, 23], but these are orthogonal to the contribution of our paper. A query over $D$ is a search keyword $q$, and returns the database row $t$ such that the attribute value $t.A$ is equal to $q$. Thus, a set of queries over $D$ can be represented by the set of its corresponding keywords, that is $Q = \{q_1, \ldots, q_n\}$. In accordance with the literature, all our sets have bag semantics, i.e. elements can be repeated.

Since the data is outsourced the client may want to encrypt its data. This can be implemented in the semi-honest model using searchable encryption, e.g. [7, 19]. In searchable encryption the client encrypts its data and builds an encrypted index that is outsourced to the server. The server learns no information from the encrypted index, but is trusted to perform queries over the index. To do so the client encrypts its search keyword to a query token and sends it to the server. The server can match the token to the encrypted index and return all ciphertexts whose plaintext matches the search keyword in the query token. Formal definitions of searchable encryption are available in the literature [7]. Since we assume that the keyword frequency has been smoothed, the access pattern does not leak frequency information, i.e. each query has the same result set size. Nevertheless, query pattern leakage remains and the problem is that the server controlled by an adversary can compute the following, very simple *query frequency attack*. Note that this attack is feasible even if the query is probabilistically encrypted, since the frequency-smoothed observed access pattern still leaks the query pattern in a somewhat static database, because the access pattern is overlapping for identical query keywords, but non-overlapping for distinct query keywords. We will present in Section 4 a query smoothing algorithm that can be combined with searchable encryption to prevent this and other attacks.

*Adversarial Model.* We consider an adversarial model in which a honest-but-curious adversary controls the server and observes the queries from the client leaking the access and query patterns. The adversary has only prior knowledge about the query and keyword distribution, but no other background information, e.g. temporal relations or partial plaintext knowledge. The adversary's goal is to recover the plaintexts of the database and query keywords.

*Query Frequency Attack.* We describe a frequency analysis attack that exploits the query pattern leakage. We determine the adversary's success of correctly recovering keywords. Assume that the adversary has access to a set of queries $Q_B = \{q_1, \ldots, q_n\}$ over $D$, referred to as background knowledge. Furthermore, we assume that the adversary observes another set of queries $Q_O = \{q_1^{'}, \ldots, q_l^{'}\}$ over $D$, issued from the client. Note that $n$ is not necessarily equal to $l$ and the frequency of keywords in each query set follows the Zipf distribution [39, 43, 45]. Given the background $Q_B$ and observed query set $Q_O$, the adversary constructs a frequency table from each query set by sorting the keywords in non-increasing order of frequency. That is, the keyword with highest frequency corresponds to rank 1, the keyword with the second largest frequency corresponds to rank 2, and so on. We denote by $T_B$ and $T_O$ the frequency tables of $Q_B$ and $Q_O$ respectively, each consisting of pairs of the form $(a_1, f_{a_1}), \ldots, (a_m, f_{a_m})$. In $T_B$, $a_i$ is a keyword and in $T_O$, $a_i$ is a query token. $f_{a_i}$ is the number of times that a keyword or token $a_i$ appears in the corresponding set. Finally, the attack works by outputting a guessed keyword $a_i \in T_B$ for the observed query token related to the keyword performed by the client $a_i \in T_O$ at the same rank. Since the background knowledge is imprecise, we report two measures of accuracy for this attack. First, we compute rank distance between equal keywords in $T_B$ and $T_O$: denote by $r_{a_i}(T_B)$ the rank of keyword $a_i$ in $T_B$, the adversary observes the corresponding rank $r_{a_i}(T_O)$ of the same keyword from $T_O$ and we compute the absolute difference of the two ranks as $d_{a_i} = \left| r_{a_i}(T_B) - r_{a_i}(T_O) \right|$. In consequence, we define the average rank distance over $m$ keywords as:

$$d_{avg}(T_B, T_O) = \frac{\sum_{i=1}^{m} d_{a_i}}{m}$$

When the difference of the two ranks $d_{a_i} = 0$, we have a correct guess for keyword $a_i$. The larger the average distance, the worse the guess of the adversary. Second, we report the number of correct guesses.

*Query Frequency Attack: Experimental Results.* To show the feasibility of the query frequency attack against different observed query sets in practice, we performed an experiment as shown in Figure 1. We chose 5 different sizes of observed query sets $10^6 \leq |Q_O| \leq 10^7$ and a background knowledge query set of size $|Q_B| = 10^7$, which both are drawn from a Zipf distribution with exponent $s = 1.001$ over $m = 16384$ number of keywords. The average rank distance decreases significantly when the size of $Q_O$ increases: for $|Q_O| = 10^6$, the average distance is $d_{avg}(Q_B, Q_O) = 1557$ that is much higher than $d_{avg}(Q_B, Q_O) = 804$ when $|Q_O| = |Q_B|$. Furthermore, it can be observed that the adversary achieves more correct guesses, i.e., keywords where the rank difference is 0, when a large set of user queries is leaked (red points in Figure 1). For a small query set $|Q_O| = 10^6$, the number of correct guesses is 76 that is less than the number of correct guesses 134 with a large query set $|Q_O| = |Q_B| = 10^7$. This is due to the fact that when the size of user's leaked queries increases, the observed frequencies of keywords become closer to

the frequencies in the background query set. As a result, the attack success rate grows with the number of observed user queries by the adversary, i.e. there is little protection against a long-lasting, persistent adversary.

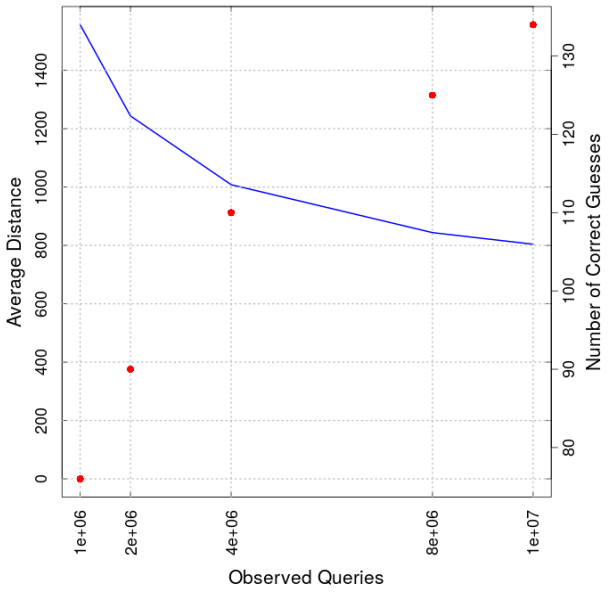


**Figure 1: The blue line depicts the average distance between guesses and the true rank over all elements for different sizes of observed sets and a fixed-size background set ($|Q_B| = 10^7$). The red points depict the number of keywords that have been guessed correctly by the adversary controlling the server, averaged over 10 runs.**

# 4 SECURE QUERY PATTERN

## 4.1 Security Objective

In order to prevent the query frequency attack and similar attacks exploiting frequency information we need to hide the frequency distribution of queries. We define the notion of a $d$-smooth query pattern.

Definition 1. *Let $d > 1$ be a positive constant. We say that a sequence $Q$ of queries is $d$-smooth, if for all keywords that belong to an attribute set $A$ there exist a set $A' \subseteq A$, such that,*

$$\forall a_i, a_j \in A' | \{a_i\} \cap Q| = |\{a_j\} \cap Q| \wedge |A'| \geq d$$

Note that our definition ensures the query frequency of at least $d$ keywords is the same and hence thwarts query frequency analysis attacks. In this aspect the of a $d$-smooth query sequence is similar to $k$-anonymity [42] that ensures that $k$ database tuples are indistinguishable in their quasi-identifying attributes. However, attacks on $k$-anonymity, e.g. background attacks [33], do not apply to $d$-smooth query sequences, since there is no plaintext sensitive attribute associated with the smooth attribute. An adversary controlling the server can only obtain approximate information about query frequencies. He will always have a set of $d$ queries he cannot distinguish based on frequency making any analysis attempt approximate.

## 4.2 Approach

In our protection algorithm we assume that the query pattern is Zipf-distributed [41]. This is a distribution commonly found in real-world query sets [39, 43, 45]. In a Zipf distribution the occurrence of the $i$-th most frequent event occurs proportional to $\frac{1}{i}$. Let $\tilde{a}_1, \ldots, \tilde{a}_m$ be the observed keywords sorted according to their query frequency distribution. Then

$$|Q \cap \{\tilde{a}_i\}| = \frac{n}{iH_m}$$

where $H_m = \sum_{i=1}^{m} \frac{1}{i}$ denotes the $m$-th harmonic number. Suppose the space of keywords is divided into $k$ sections. The first section has size $d$ and contains the most frequent keywords $\tilde{a}_1, \ldots, \tilde{a}_d$. The second section has size $2d$ and contains the keywords $\tilde{a}_{d+1}, \ldots, \tilde{a}_{3d+1}$. The third section has size $4d$, the fourth $8d$ and so forth. We create a local cache of keywords on the client. The size of the cache will be kept small ($O(1)$) and the cache will be queried by the client before issuing a query to the server. Nevertheless, even in case an element is in the cache the client may issue a query to the server to smooth the query pattern (see details in Section 4.4 and Algorithm 1). The cache is pre-loaded and the $i_{cache}$ most frequent keywords in each section are stored in this cache. Let $f_j$ be the number of queries of the first keyword in section $j$ not in the cache. Each keyword in the cache will be queried $f_j$ times from the server and else be read from the cache. Each keyword in section $j$ not in the cache will be padded to $f_j$ queries by using fake queries. Hence, we select fake queries according to the following distribution

$$Pr[\tilde{a}_i] = \frac{\max(0, f_j - |Q \cap \{\tilde{a}_i\}|)}{\sum_{i=1}^{m} \max(0, f_j - |Q \cap \{\tilde{a}_i\}|)} = \frac{\max(0, f_j - |Q \cap \{\tilde{a}_i\}|)}{\mathcal{N}}$$

where $Pr[\tilde{a}_i]$ is the probability of generating a fake query for keyword $a_i$ located in section $j$ and $\mathcal{N}$ is the total number of fake queries. The resulting query distribution looks as in Figure 2.

Note that the exponentially increasing section sizes ensure that at least half of the keywords will have the same query frequency after smoothing, even for very small $d$. In consequence, while high frequency queries are protected in a set of size at least $d$, low frequency queries are protected in a much larger set. Islam et al. [22] note in their attack that low frequency keywords carry the most information to distinguish plaintexts using only the query and access pattern. Furthermore, we show experimentally in Section 5 that reasonably large values of $d$ are practically feasible.

In our query smoothing algorithm we assume that the querier has knowledge of the query distribution and that the query distribution is static over a fixed period of $n$ queries. These are common assumptions in data management, e.g. also made in [11, 37]. The querier can collect the query distribution over a period of time and then use this distribution as background knowledge for protection in the subsequent period while collecting a new query distribution in this period. As long as the distributions from two consecutive periods are somewhat similar, our scheme provides good performance and reasonable protection.

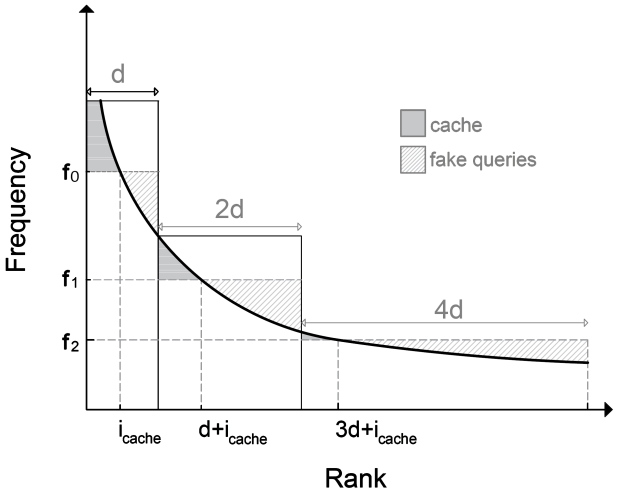


**Figure 2: Sample query distribution with three sections resulting from our query smoothing algorithm.**

### 4.3 Theoretical Analysis of the Overhead

In this section we present a theoretical analysis of the number of fake queries with regard to the cache size and security parameter. We prove that our algorithm has a constant overhead of fake queries per real query and a constant cache size for a given security parameter $d$.

Theorem 1. *For a security parameter $d$ in a $d$-smooth query pattern, the overhead (the number of fake queries per real query) and the cache size is $O(1)$, i.e. constant in the database size $m$.*

Proof. Let $m$ be the number of keywords, $i$ be the index of a keyword sorted in non-increasing order of frequency and suppose that $s$ is the exponent value characterizing the Zipf distribution. Then the frequency of the $i$-th keyword out of a population of $m$ is defined as:

$$f(i,s,m) = \frac{1}{i^s H_m}$$

Now let $Q$ be a set of real queries of size $n$ and let $k$ denote the number of sections. Let $d$ be the security parameter, i.e. the size of first section in our definition of a $d$-smooth query pattern. To simplify our analysis in the following proof, we assume that $s = 1$ and $d$ is a multiple of 2. Recall that $i_{cache}$ is the index of the first keyword that is not in the cache.

Following the above assumptions, for a given section $j = 0, 1, \ldots, k-1$, we can define $f_j$ as the frequency of the $i_{cache}$-th most frequent keyword that is

$$f_j = \frac{n}{((2^j-1)d + i_{cache})H_m}$$

$i_{cache}$ is then computed as

$$i_{cache} = \frac{n}{f_j H_m} - (2^{j-1}-1)d$$

Let $\mathcal{N}_j$ be the total number of fake queries for section $j$. $\mathcal{N}_j$ is given by

$$\begin{aligned}\mathcal{N}_j &= \sum_{i=(2^j-1)d+i_{cache}}^{(2^{j+1}-1)d} (f_j - \frac{n}{iH_m}) \\ &= \Big(2^j d - i_{cache} + 1\Big) f_j \\ &\quad - \Big(\frac{n}{H_m}(H_{(2^{j+1}-1)d} - H_{(2^j-1)d+i_{cache}-1})\Big).\end{aligned}$$

Thus, we can write the total number $\mathcal{N}$ of fake queries for $k$ sections

$$\begin{aligned}\mathcal{N} &= \sum_{j=0}^{k-1} \Big[(2^j d - i_{cache} + 1) f_j \\ &\quad - \Big(\frac{n}{H_m}(H_{(2^{j+1}-1)d} - H_{(2^j-1)d+i_{cache}-1})\Big)\Big] \\ &= d\sum_{j=0}^{k-1} 2^j f_j - (i_{cache}-1)\sum_{j=0}^{k-1} f_j \\ &\quad - \frac{n}{H_m}\sum_{j=0}^{k-1}(H_{(2^{j+1}-1)d} - H_{(2^j-1)d+i_{cache}-1}).\end{aligned} \tag{1}$$

If we only focus on the first term of Eq. (1), the total number of fake queries is smaller than $d\sum_{j=0}^{k-1} 2^j f_j$, where

$$\begin{aligned}\sum_{j=0}^{k-1} 2^j f_j &= \sum_{j=0}^{k-1} \frac{2^j n}{(2^j-1)d + i_{cache})H_m} \\ &= \frac{n}{i_{cache}H_m} + \frac{n}{H_m}\sum_{j=1}^{k-1}\frac{2^j}{(2^j-1)d + i_{cache})}.\end{aligned} \tag{2}$$

We now investigate the summation in the second term of Eq. (2):

$$\sum_{j=1}^{k-1}\frac{2^j}{(2^j-1)d + i_{cache}} = \sum_{j=1}^{k-1}\frac{1}{d + (i_{cache}-d)\frac{1}{2^j}}. \tag{3}$$

The value of $(i_{cache} - d)\frac{1}{2^j}$ is negative, since $i_{cache} < d$. Thus

$$(i_{cache}-d)\frac{1}{2^j} \geq (i_{cache}-d)\frac{1}{2}.$$

And therefore we can write Eq. 3 as

$$\sum_{j=1}^{k-1}\frac{1}{d + (i_{cache}-d)\frac{1}{2^j}} < \frac{2(k-1)}{i_{cache}+d}.$$

Note that since $i_{cache} > 0$

$$\sum_{j=1}^{k-1}\frac{1}{d + (i_{cache}-d)\frac{1}{2^j}} < 2(k-1)d < 2kd.$$

Thus

$$\mathcal{N} < d\Big(\frac{n}{H_m i_{cache}} + \frac{2kn}{dH_m}\Big)$$

Consequently, the number of fake queries per real query is

$$\frac{\mathcal{N}}{n} < \frac{d}{H_m i_{cache}} + \frac{2k}{H_m}$$

We set the number of fake queries per real query equal to the total cache size, since we aim for constant number and size for both. I.e.,

if we obtain a constant bound on one, we have a constant bound on the other. Then we have

$$\frac{d}{H_m i_{cache}} + \frac{2k}{H_m} = k i_{cache}. \tag{4}$$

Turning Eq. 4 into a quadratic equation of $i_{cache}$, we have

$$i_{cache}^2 - \frac{2}{H_m} i_{cache} - \frac{d}{kH_m} = 0. \tag{5}$$

Finally, solving Eq. 5 yields the desired $i_{cache}$:

$$i_{cache} = \frac{1}{H_m} + \sqrt{\frac{1}{H_m^2} + \frac{d}{kH_m}} = O\left(\sqrt{\frac{d}{kH_m}}\right)$$

The number of fake queries per real query and the total cache size is hence

$$\frac{\mathcal{N}}{n} = k i_{cache} = O(\sqrt{\frac{dk}{H_m}})$$

If we choose $d = O(1)$ constant, then $k = O(\log m)$ is logarithmic in the database size and since $H_m = \Theta(\log n)$, the total cache size is also $O(1)$ constant and the number of fake queries per real query $\frac{\mathcal{N}}{n}$, i.e. the overhead of our scheme, is $O(1)$ constant.

$$d = O(1) \Longrightarrow \frac{\mathcal{N}}{n} = k i_{cache} = O(1)$$

□

$k i_{cache} = O(1)$ implies that while the number $k$ of section increases, the cache size per section decreases, but their sum stays constant. In our allocation strategy of caches, this means that the higher numbered section may not even have a cache.

## 4.4 Algorithm

The final query smoothing algorithm is presented in Algorithm 1. It chooses $\frac{\mathcal{N}}{\mathcal{N}+n} + 1 = O(1)$ queries and permutes them before sending them to the server. Note that the expected probability of the coin flip $\gamma_3$ in line 30 is constant.

$$E[Pr[\gamma_3 = 1]] = \frac{\mathcal{N}}{\mathcal{N}+n} = \frac{O(1)}{O(1)+1}$$

Hence the while loop from lines 26–32 is expected to terminate in a constant number of rounds and the expected runtime complexity of Algorithm 1 is $O(1)$.

**Algorithm 1** Fake Query Generation

1: **function** Query($q$)
2: /* $W = \{a_1, \ldots, a_m\}$ is a set of available keywords */
3: /* $f : W \to \mathbb{N}$ is the observed, real keyword frequency */
4: /* $\tilde{a}_i \in W$ */
5: /* $i < i' \Longrightarrow f(\tilde{a}_i) \geq f(\tilde{a}_{i'})$ */
6: /* $f_j$ is the target frequency of a keyword in section $j$ */
7: /* $\mathcal{N}$ is the total number of fake queries */
8: /* $d$ is the security parameter */
9: /* $C$ is a map used as the cache */
10: /* $\pi$ is a random permutation function */
11: /* $S$ is a multi-set of queries over $W$ */
12: /* $q \in W$ */
13: $S \leftarrow \emptyset$
14: **if** $q \in C$ **then**
15: Lookup $i$, such that $\tilde{a}_i = q$
16: $j \leftarrow \lfloor \log \frac{i}{d} \rfloor$
17: Lookup $f_j, f(\tilde{a}_i)$
18: Flip coin $\gamma_1$ with $Pr[\gamma_1 = 1] = \frac{f_j}{f(q)}$
19: **if** $\gamma_1 = 1$ **then**
20: $S \leftarrow S \cup \{q\}$
21: **else**
22: $S \leftarrow S \cup \{q\}$
23: Flip coin $\gamma_2$ with $Pr[\gamma_2 = 1] = \frac{\mathcal{N}}{n} - \lfloor \frac{\mathcal{N}}{n} \rfloor$
24: **if** $\gamma_2 = 0$ **then**
25: $S \leftarrow S \cup \{dummy\}$
26: **while** $|S| < \frac{\mathcal{N}}{n} + 1$ **do**
27: Uniformly choose $i \xleftarrow{\$} [1, m]$
28: $j \leftarrow \lfloor \log \frac{i}{d} \rfloor$
29: Lookup $f_j, f(\tilde{a}_i)$
30: Flip coin $\gamma_3$ with $Pr[\gamma_3 = 1] = \frac{\max(0, f_j - f(\tilde{a}_i))}{\mathcal{N}/m}$
31: **if** $\gamma_3 = 1$ **then**
32: $S \leftarrow S \cup \{\tilde{a}_i\}$
33: $S \leftarrow \pi(S \setminus \{dummy\})$
34: Send result query $S$ to server, each query at a time

# 5 IMPLEMENTATION

## 5.1 Security Evaluation

In Section 3, we presented a query frequency analysis attack that used background information about the query distribution. Given this auxiliary information, we showed experimentally that an adversary controlling the server is able to infer query keywords leaked by the query frequency with high accuracy.

In this section, we investigate the effectiveness of our query smoothing algorithm as a countermeasure to defeat this attack. Note that the query frequency analysis is the most simple, yet most effective attack exploiting the frequency leakage of the query pattern. Hence we conjecture that preventing this attack implies preventing other attacks that exploiting the same leakage. Since our query smoothing algorithm transforms the observed query distribution to a uniform distribution by interspersing each real query with a number of fake queries, the adversary will not observe the true frequency of real queries. Thus, the observed frequency of each keyword is distorted in the attack and indistinguishable from at least $d - 1$ other keywords. Hence frequency analysis – the most effective attack method [29] – has limited accuracy and is limited to a guess among the at least $d$ keywords.

To validate that our query smoothing algorithm protects against leakage-abuse attacks we ran the query frequency attack from Section 3 again and obtained the results depicted in Figure 3. As before we assume that the adversary has a background query set $|Q_B| = 10^7$ that is drawn from a Zipf distribution with specified exponent $s = 1.001$ over $m = 16384$. We again choose 5 observed query sets with $10^6 \leq |Q_O| \leq 10^7$ real queries. Then for each $Q_O$, we intersperse fake queries using our smoothing algorithm for sections $k = 1, \ldots, 4$. This increases the size of each observed query set to $Q_O^{'}$ such that $|Q_O^{'}| \gg |Q_O|$. Finally, we computed average rank distance $d_{avg}(Q_O^{'}, Q_B)$ as defined in Section 3. We can see from Figure 3 that our query smoothing algorithm achieves a protection level that withstands the query frequency attack. The average distance is always higher and does not decrease with an increasing observed query set size. Furthermore, the number of correct guesses by the adversary is very small. When $|Q_O| = 10^6$, $d_{avg}(Q_B, Q_O^{'}) = 5450$ with $k = 1$ and the number of correct guesses is only 2. Similar numbers can be observed for a large query set $|Q_B| = |Q_O| = 10^7$ with the same number of sections. Obviously, when $k$ increases the

average distance decreases in Figure 3. For example, with $k = 4$ and $|Q_O| = 10^7$, the average distance $d_{avg}(Q_B, Q_O^{'}) = 2093$, and the number of correct guesses is 4, yet that is still not a significant advantage for the adversary compared to 134 correct guesses and $d_{avg}(Q_B, Q_O) = 804$ without our query smoothing algorithm. Thus, our results underpin the effectiveness of our protection model to thwart the query frequency analysis attack.

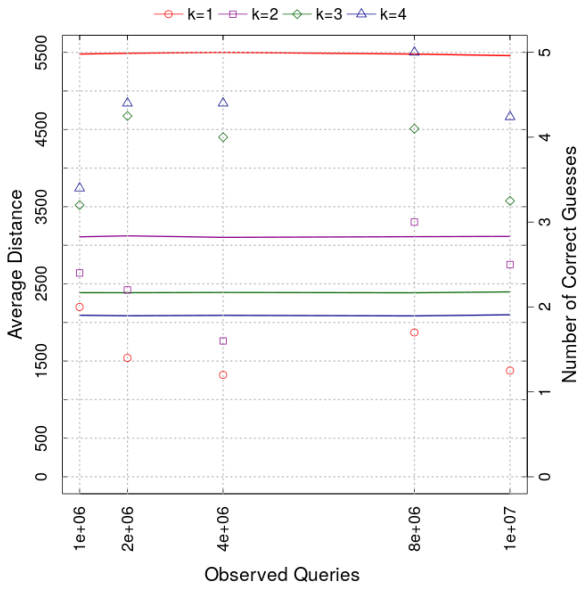


**Figure 3: The average distance between guesses and true keyword ranks over different observed query set sizes ($10^6 \leq |Q_O| \leq 10^7$) and a fixed-size background set ($Q_B = 10^7$) with our query smoothing algorithm employed. Different lines depict different numbers of sections: $k = 1$ to $k = 4$. The colored symbols represent the number of keywords that have been guessed correctly by the adversary controlling the server, averaged over 10 runs.**

## 5.2 Performance Evaluation

We experimentally measured the performance of a searchable encryption scheme protected by our query smoothing algorithm in comparison to an ORAM implementation. We next describe our parameter choices and show how our scheme scales for large database sizes when compared to an ORAM-based solution.

*Setup.* Two experimental results are compared: a dynamic symmetric searchable encryption (*DSSE*) scheme by Cash et al. [6] protected by our fake query generation algorithm and a simple search over ORAM by Stefanov [44]. We use the same libraries (implementation) also used in related work comparing the same cryptographic schemes [10, 21]. We evaluated the performance of both approaches on a 64-bit Ubuntu machine with AMD phenom(tm) II X4 955 Processor and 8GB RAM. Moreover, we used a block size $B = 4096$ bits and security parameter of 128 bits.

*DSSE.* We used the Java implementation of the most efficient *DSSE* algorithm namely $\Pi_{2Lev}$ that is available in the Clusion library [2] running over MongoDB for comparable results. We used 10 documents of keywords drawn uniformly such that each keyword appears at least in one document. In this experiment, we measured the total wall clock time required by the server to retrieve the document identifiers associated with a set of keyword queries including both – real and fake ones.

*PathORAM.* We performed our ORAM-based experiment by choosing the C++ implementation of PathORAM, available in SEAL-ORAM library [1]. The difference in programming languages between *DSSE* and PathORAM should improve the relative performance of ORAM. We measured the computation time necessary to retrieve the blocks for each query containing only the real queries. We used the simplest form of query where we queried $i$ for keyword $a_i$. Note that in any practical searchable encryption scheme, one would first have to query an index which can potentially increase the overhead even superlinearly [34]. We set $Z = 4$, which gives the number of blocks for each bucket on the server.

The results on query performance for varying database sizes $2^{10} \leq m \leq 2^{16}$ are depicted in Figure 4. For a set of $n = 10^6$ real queries, we observed that the PathORAM search time grows when the size of database increases as expected. While the time for searchable encryption is constant using $\Pi_{2Lev}$ with a set of queries containing $10^6$ real queries and a variable number of fake queries. Note that the number of fake queries varies depending on the size of database and the number of sections. For $\Pi_{2Lev}$, we computed the search time of sections 1 to 4 for $2^{10} \leq m \leq 2^{16}$, respectively, i.e. using a constant overhead as computed in Section 4.3. For $m = 65536$ and $k = 7$ the $\Pi_{2Lev}$ on a query set takes 1710 seconds while PathORAM takes 46970 seconds using the same database size. As a result, in practice our $\Pi_{2Lev}$ using real and fake queries achieves up to 27× improvement in performance compared to the most simple PathORAM construction while still hiding the query pattern. In conclusion, we can confirm that the performance of our query smoothing algorithm in combination with $\Pi_{2Lev}$ has a constant overhead as we proved in Section 4.3, while ORAM has at least a logarithmic overhead in the database size. Furthermore, the practical performance of our approach is much better than that of ORAM.

## 5.3 Overhead Evaluation

In our analysis in Section 4.3 we proved that the number of fake queries per real query is constant for a given security parameter $d$. Here, we present an experiment to show that this fact can also be observed in practice for different, large database sizes. We examined the ratio of total fake queries over real queries for different database sizes and different, but constant security parameters $d$. We chose the parameters sets $2^{17} \leq m \leq 2^{19}$, $n = 10^7$ real queries, and $d \in \{256, 512\}$. The reported results are averaged over 100 runs and shown in Figure 5. We set the initial database size to $m = 2^{17}$ and then observe the total number of fake queries over the total number of real queries doubling $m$ and increasing the number of sections $k$ by 1 at the same time. The red and blue lines correspond to $d = 256$ and $d = 512$, respectively. The first point $m = 2^{17}$ on the blue line depicts the ratio $\frac{N}{n}$ for $k = 8$ sections, while the

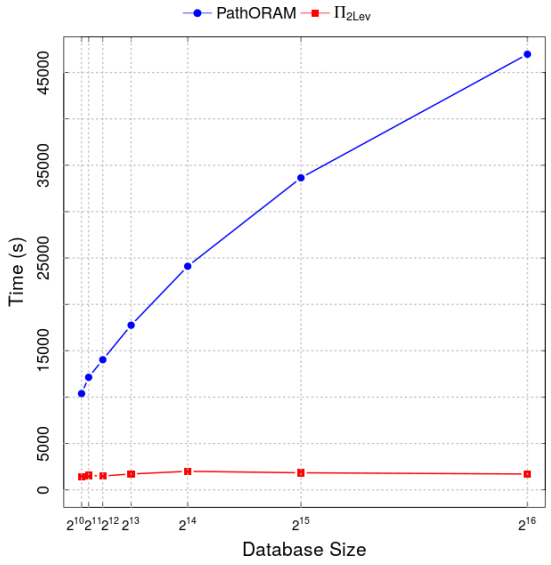


**Figure 4: The red line with squares depicts the search time of our query smoothing algorithm with security parameter $d = 512$ combined with $\Pi_{2Lev}$ to retrieve document identifiers (for $n = 10^6$ real queries). The number of sections starts at $k = 1$ and increments as the database size doubles. The blue line depicts the time needed for *PathORAM* to retrieve blocks for a set of $n = 10^6$ real queries.**

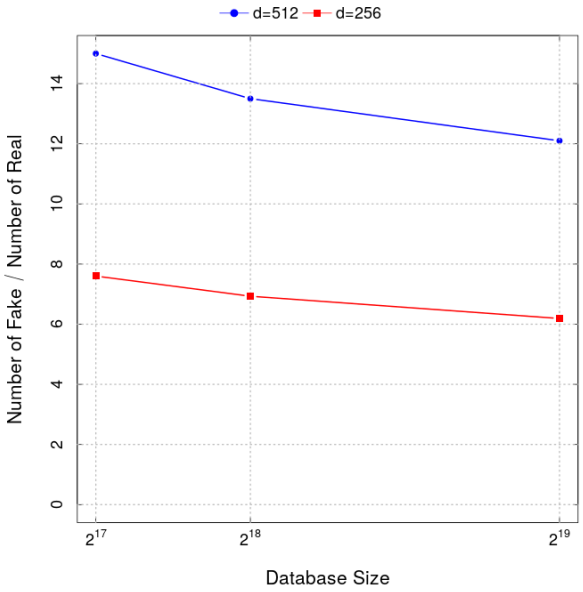


**Figure 5: The ratio of total number of fake queries over $n = 10^7$ real queries with security parameters $d = \{256, 512\}$, averaged over 100 runs.**

corresponding point on the red line has $k = 9$. We observe the line slowly approaching a constant overhead of $\frac{N}{n}$ for both security parameters. Indeed for $m = 2^{18}$, the ratio $\frac{N}{n} = 13.80$ and for $m = 2^{19}$ the ratio is 12.60. As a result, we can confirm that $\frac{N}{n}$ is constant independent of database size. To better understand the behavior of our query smoothing algorithm, we performed an experiment with different sections, $k = 1, \ldots, 7$, while setting the number of real queries to $10^7$, and the database size to $2^{10} \leq m \leq 2^{15}$. The goal is to analyze the ratio $\frac{N}{n}$ as we increase the database size $m$ for a given number of sections $k$. Clearly, when $k$ increases the ratio $\frac{N}{n}$ decreases in Figure 6. When $m$ increases and $k$ is fixed, the ratio $\frac{N}{n}$ increases with a decreasing derivative, i.e. the increase is lower for larger database size. This suggests that one can maintain large security parameters $d$ even for large database sizes $m$.

## 6 RANGE QUERIES

We now turn from keyword queries to range queries. Our $d$-smooth query scheme can be implemented with range-searchable encryption scheme that does not have static leakage, e.g. [9, 20, 46]. Instead, of $m$ possible queries there are now $O(m^2)$ possible queries. However, this information may be stored in a compressed form. For example, one distribution may characterize the starting point of a query and another, independent distribution the length of the range query. The database still contains $m$ elements, each of which returns a fixed-size result when queried.

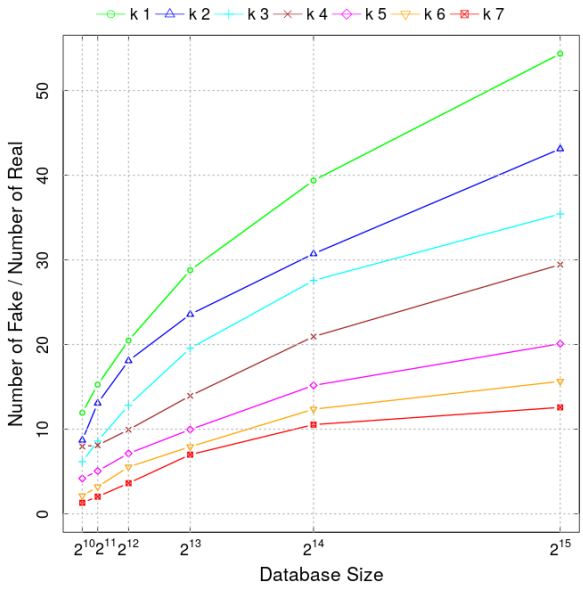


**Figure 6: The ratio of total number of fake queries over $n = 10^7$ real queries for different numbers of sections $k = 1, \ldots, 7$, averaged over 35 runs.**

It has been observed that the leakage from range queries is more detrimental than that from keyword queries. Recent attacks on encrypted range queries [13, 14, 17, 24, 28] demonstrate that even with a few, i.e. linear in the database size, range queries one can

reasonably reliably reconstruct large parts of the database plaintext. These results put into question the feasibility of efficient, secure range queries over encrypted data. Hence, it is important to devise protection techniques for secure range queries.

## 6.1 Overview of Attacks

First, we briefly review the generic attack [24] which served as a blueprint for all subsequent attacks that improve its efficiency. At a high level, this generic attack operates as follows: For a database of $n$ records, each with a unique keyword drawn from a larger range $R = [1, \ldots, N]$ $(N > n)$, the scheme retrieves the order of records by sampling enough queries drawn from a uniform distribution, i.e. independent of the keyword distribution. Its first step is to determine a guess for the minimal (or maximal) position of a record, i.e., a position with value 1 or $N$. Once this position is identified as the keyword least frequently queried, we assume that it corresponds to value 1, i.e. the algorithm takes the order $I_1$ to be the symmetric difference of all sets. Given $I_1$, it determines $I_2$ by searching the query results for the smallest proper superset of $I_1$. Similarly, position $I_j$ can be determined by searching the query results for the smallest proper smallest proper subset of $I_1, \ldots, I_{j-1}$. Thus, based on a simple analysis on Chernoff bounds [24], the author showed that after observing the required number of $N^4$ queries, where $N$ is the domain size, their attack successfully recovers the plaintexts of the entire database. This requires only $O(n^2)$ number of queries for dense databases.

A crucial assumption in this attack is that the each range query $[a, b]$ is drawn from the set of queries where $a \leq b$. In reverse, it assumes that a range $[a, b]$ where $a > b$ is never queried. This assumption has dire consequences on the security of any range-searchable scheme with access pattern leakage. In particular, if the endpoints of the query are drawn uniformly or from a Zipf distribution, elements near the middle of the order over the domain are much more likely to be queried than elements near the ends of the order. This implies that the frequency of access is directly proportional to the rank of the element and that the rank of an element can be derived from its access frequency. Knowledge of this distribution is exploited by all attacks on range queries [13, 14, 17, 24, 28]. The attack above uses it to reconstruct the first or last element in the order.

In this paper we do not question the validity of this assumption in practice, albeit it is not guaranteed, but show that even if it is true, one can deploy successful countermeasures.

## 6.2 Fake Queries as a Countermeasure

We now show how to protect against the order leakage from overlapping range queries. We again assume that the querier has knowledge of its own query distribution as also assumed in many works in data management, e.g. [11, 37]. This distribution does not contain any queries $[a, b]$ where $a > b$, but it contains a frequency for all queries $a \leq b$. Note that in order to smooth the query distribution it is necessary that all ranges, including those where $a > b$, are feasible queries. Assume only ranges where $a \leq b$ are queried. Then keywords belong to a different number of ranges depending on their rank. If the frequency of queries per keyword is smooth, the frequency of queried ranges is not. Vice versa, if the frequency of queried ranges is smooth, the frequency of queries per keywords is not. This conflict can be broken by including ranges where $a > b$ as feasible queries.

A range query with $a > b$ may be interpreted as two range queries of the form where $a \leq b$. Let $min$ be the minimum element in the order of the domain and $max$ be the maximum element in the order of the domain. Then a range query $[a, b]$ where $a > b$, may be written as $[min, b] \cup [a, max]$. Range covers can be easily extended to these ranges and hence most range-searchable encryption schemes, including [9, 20], can use a single search token to implement a query of the form $[min, b] \cup [a, max]$. Hence these query tokens are indistinguishable from real range query tokens to an adversary controlling the server.

Let $Q$ be the query distribution for ranges $r = [a, b]$ where $a \leq b$. $Q$ consists of pairs $r, f(r)$ where $f(r)$ is the frequency of each range. We assume that $Q$ is sorted in descending order of the frequency, i.e. $\forall i, j \; i \geq j \iff f(r_i) \geq f(r_j)$. Let $Q'$ be the set of ranges $r = [a, b]$ where $a > b$. The querier randomly intersperses the ranges of $Q'$ among the distribution $Q$, i.e. for each range $r \in Q'$ uniformly randomly select a rank $i$ $(0 \leq i \leq \frac{m^2}{2})$ and insert $r$ after rank $i$ (or before rank 1 in case of $i = 0$). Let $Q_R$ be the resulting query distribution. Note that $Q_R$ contains ranges at a rank $i$, such that there exists a range at a rank $j > i$ where $f(r_j) > 0$, but $f(r_i) = 0 < f(r_j)$.

The querier again divides this distribution into $k$ sections and computes the number of queries $f_j$ for each section under our protection scheme. Given a range query as in Algorithm 1 the querier now chooses fake queries according to the distribution

$$Pr[r_i] = \frac{\max(0, f_j - |Q_R \cap \{r_i\}|)}{\sum_{i=1}^{m^2} \max(0, f_j - |Q_R \cap \{r_i\}|)}$$

Let $Q_F$ be the fake queries drawn according to the above distribution. The adversary controlling the server observes the query set $Q_P = Q_R \cup Q_F$. We conjecture that $Q_P$ results in a uniform access pattern distribution over all $m$ elements $a_i$ in the database. We empirically confirm this conjecture in the next Section.

We now explain how a uniform access pattern distribution thwarts the attacks [13, 14, 17, 24, 28]. Clearly, the access frequency no longer correlates with the rank of the element and hence this cannot be exploited directly. In the generic attack [24] described above this prevents the first step of identifying the two endpoints in the keyword domain.

Furthermore, the intersection relation of two accesses can also no longer be used to determine the rank of an element. To see this, we observe the effect of range queries where $a > b$. Since these queries may "wrap around" the endpoints of the domain, intersecting result sets are no longer necessarily sorted, i.e. non-intersecting parts of the range may be smaller or larger than the intersection. Although two ranges may now have two distinct regions of intersection indicating that one range is a fake one, the adversary cannot determine which one of the two is the fake one and hence this does not help in reconstructing the rank of an element. The adversary can still arrange the elements, since intersecting results sets are adjacent, but the arrangement is only a random rotation of the order. Hence, all attacks that rely on the intersection relation of range queries to reconstruct the order are also prevented.

In summary, in combination with the ciphertext frequency smoothing techniques of [22, 23] our technique of smoothing the query frequency distribution using fake queries can prevent all known reconstruction attacks on range-searchable, encrypted databases [13, 14, 17, 24, 28]. This results in a searchable encryption schemes that can protect against attacks that not even ORAM by itself can protect against. However, we emphasize that our scheme is not leakage-free and reveals information, such as a random rotation of the order, which could be used in future attacks. Furthermore, as we have shown in Section 5.2 our scheme is an order of magnitude faster than ORAM and scales better with increasing database size.

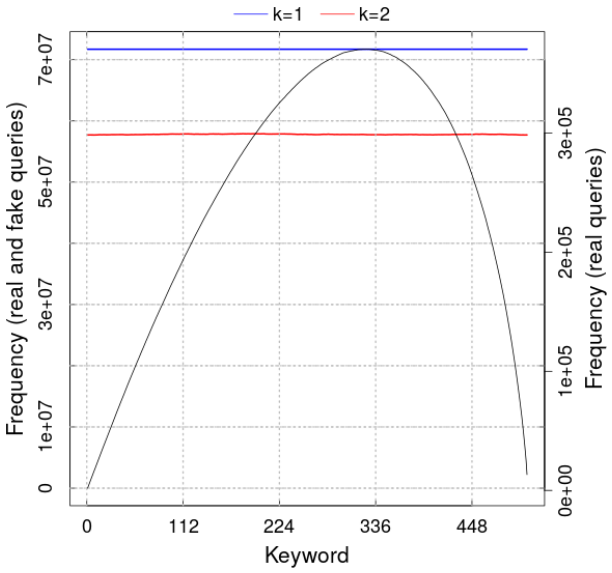


**Figure 7: The frequency corresponding to $2^9$ keywords observed in a query set with $10^6$ real range queries of the form $[a, b]$ $(a \leq b)$. The black line depicts each keyword frequency observed in a query set drawn uniformly randomly with $a \leq b$ . The blue and red lines depict the frequency of each keyword observed after interspersing fake queries including queries where $a > b$ with numbers of sections $k = 1$ and $k = 2$, respectively.**

## 6.3 Evaluation

We evaluate whether our query smoothing algorithm prevents order leakage from the access pattern frequency and hence the attacks on encrypted range queries. We implemented the algorithm described in Section 6.2. We executed it over a database with $2^9$ keywords which results in $2^{18} \approx 3 \cdot 10^5$ possible ranges. We uniformly randomly selected an order for the Zipf distribution of the start point of the range and uniformly randomly selected an order for the Zipf distribution of the length of the range. Using these two distributions we used rejection sampling to draw $10^6$ ranges of the form $a \leq b$, i.e. we discard samples of the form $a > b$ and redraw the length of the range. We expect the distribution of queried ranges to be smooth. However, the distribution of queries per keyword is not smooth as depicted by the black line in Figure 7.

Then we used our query smoothing algorithm in order to smooth the range queries using fake queries drawn as above, i.e. including ranges where $a > b$. We did this for $k = 1$ (blue line) and $k = 2$ (red line) sections and the distribution of queries per keyword is now also smooth. We can see that the red line is lower than the blue as expected, since with more sections the number of fake queries per real query decreases. We conclude that our query smoothing algorithm results in an access frequency for each keyword that is approximately the same, i.e. smooth, and hence prevents attacks abusing this leakage.

# 7 CONCLUSION

In this paper we presented a query frequency smoothing algorithm. This algorithm can hide the frequency information in the query distribution by interspersing fake queries and we give a formal definition of the security achieved. Furthermore, we show theoretically and empirically that our frequency smoothing algorithm incurs a constant overhead on Zipf distributed queries, i.e. there is a constant number of fake queries for each real query independent of the database size. We also show that this number is low in practice, i.e. between 7 and 13 for large security parameters. Furthermore, we show that our algorithm not only scales better than ORAM but outperforms it in practice by an order of a magnitude. We then evaluate the practical security against the most effective, yet simple attack, and show that it provides reasonable protection.

Lastly, we show that our algorithm can be used to protect range-searchable encryption. Our query smoothing algorithm can be combined with most range-searchable encryption schemes and we show that our algorithm prevents all known reconstruction attacks on encrypted range queries when applied properly. This results in the most efficient range-searchably encrypted database that withstands all known leakage-abuse attacks.

# 8 ACKNOWLEDGMENTS

We gratefully acknowledge the support of NSERC for grants RGPIN-05849, CRDPJ-531191, IRC-537591 and the Royal Bank of Canada for funding this research.

# REFERENCES

[1] 2016. The SEAL-ORAM library. https://github.com/InitialDLab/SEAL-ORAM.
[2] 2017. The Clusion library. https://github.com/encryptedsystems/Clusion.
[3] Adi Akavia, Dan Feldman, and Hayim Shaul. 2018. Secure Search on Encrypted Data via Multi-Ring Sketch. In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS '18)*. 985–1001.
[4] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel N. Kho, and Jennie Rogers. 2017. SMCQL: Secure Query Processing for Private Data Networks. *PVLDB* 10, 6 (2017), 673–684.
[5] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-Abuse Attacks Against Searchable Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. 668–679.
[6] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *Proceedings of the 21st Network and Distributed System Security Symposium*.
[7] R. Curtmola, J. Garay, Seny Kamara, and R. Ostrovsky. 2006. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *13th ACM Conference on Computer and Communications Security (CCS '06)*. 79–88.
[8] Sabrina De Capitani Di Vimercati, Sara Foresti, Stefano Paraboschi, Gerardo Pelosi, and Pierangela Samarati. 2015. Shuffle Index: Efficient and Private Access to Outsourced Data. *ACM Trans. Storage* 11, 4, Article 19 (Oct. 2015), 19:1–19:55 pages.
[9] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. 2016. Practical Private Range Search Revisited.

In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. 185–198.

[10] Ioannis Demertzis, Rajdeep Talapatra, and Charalampos Papamanthou. 2018. Efficient Searchable Encryption Through Compression. *Proc. VLDB Endow.* 11, 11 (July 2018), 13.

[11] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning Database Configuration Parameters with iTuned. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1246–1257.

[12] Oded Goldreich and Rafail Ostrovsky. 1996. Software Protection and Simulation on Oblivious RAMs. *J. ACM* 43, 3 (May 1996), 431–473.

[13] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2019. Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks. *IACR Cryptology ePrint Archive* 2019 (2019), 11.

[14] Paul Grubbs, Marie-Sarah Lacharite, Brice Minaud, and Kenneth G. Paterson. 2018. Pump Up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. 315–331.

[15] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. 2016. Breaking Web Applications Built On Top of Encrypted Data. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. 1353–1364.

[16] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. 2017. Leakage-Abuse Attacks against Order-Revealing Encryption. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy*. 655–672.

[17] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. 2019. Encrypted Databases: New Volume Attacks against Range Queries. In *Proceedings of the 2019 ACM Conference on Computer and Communications Security CCS*.

[18] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over Encrypted Data in the Database-service-provider Model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, Wisconsin) *(SIGMOD '02)*. 216–227.

[19] Florian Hahn and Florian Kerschbaum. 2014. Searchable Encryption with Secure and Efficient Updates. In *Proceedings of the 2014 ACM Conference on Computer and Communications Security, Scottsdale*. 310–320.

[20] Florian Hahn and Florian Kerschbaum. 2016. Poly-Logarithmic Range Queries on Encrypted Data with Small Leakage. In *Proceedings of the 2016 ACM Workshop on Cloud Computing Security*. 23–34.

[21] Thang Hoang, Attila A. Yavuz, F. Betül Durak, and Jorge Guajardo. 2018. Oblivious Dynamic Searchable Encryption on Distributed Cloud Systems. In *Proceedings of the 32nd IFIP Data and Applications Security and Privacy Conference DBSec*.

[22] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation.. In *Proceedings of the 19th Network and Distributed System Security Symposium*.

[23] Seny Kamara and Tarik Moataz. 2019. Computationally Volume-Hiding Structured Encryption. In *Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT*. 183–213.

[24] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. 1329–1340.

[25] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2017. Accessing Data while Preserving Privacy. *CoRR* abs/1706.01552 (2017). arXiv:1706.01552 http://arxiv.org/abs/1706.01552

[26] Florian Kerschbaum. 2015. Frequency-Hiding Order-Preserving Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. 656–667.

[27] Florian Kerschbaum and Anselme Tueno. 2019. An Efficiently Searchable Encrypted Data Structure for Range Queries. In *Proceedings of the 24th European Symposium on Research in Computer Security*.

[28] M. Lacharite, B. Minaud, and K. G. Paterson. 2018. Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*. 297–314. https://doi.org/10.1109/SP.2018.00002

[29] Marie-Sarah Lacharité and Kenneth G. Paterson. 2015. A note on the optimality of frequency analysis vs. $\ell_p$-optimization. *IACR Cryptology ePrint Archive* 2015 (2015), 1158.

[30] Marie-Sarah Lacharité and Kenneth G. Paterson. 2018. Frequency-smoothing encryption: preventing snapshot attacks on deterministically encrypted data. *IACR Trans. Symmetric Cryptol.* 2018, 1 (2018), 277–313.

[31] Kasper Green Larsen and Jesper Buus Nielsen. 2018. Yes, There is an Oblivious RAM Lower Bound!. In *Proceedings of the 38th International Cryptology Conference CRYPTO*. 523–542.

[32] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-An Tan. 2014. Search Pattern Leakage in Searchable Encryption: Attacks and New Construction. *Inf. Sci.* 265 (May 2014), 176–188. https://doi.org/10.1016/j.ins.2013.11.021

[33] D. J. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J. Y. Halpern. 2007. Worst-Case Background Knowledge for Privacy-Preserving Data Publishing. In *Proceedings of the 23rd IEEE International Conference on Data Engineering*. 126–135.

[34] Muhammad Naveed. 2015. The Fallacy of Composition of Oblivious RAM and Searchable Encryption. *IACR Cryptology ePrint Archive* 2015 (2015), 668.

[35] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. 644–655.

[36] Simon Oya and Florian Kerschbaum. 2020. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. *CoRR* abs/2010.03465 (2020). arXiv:2010.03465 https://arxiv.org/abs/2010.03465

[37] Oguzhan Ozmen, Kenneth Salem, Jiri Schindler, and Steve Daniel. 2010. Workload-aware Storage Layout for Database Systems. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. 939–950.

[38] Giuseppe Persiano and Kevin Yeo. 2019. Lower Bounds for Differentially Private RAMs. In *Proceedings of the 38th International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT*. 404–434.

[39] Casper Petersen, Jakob Grue Simonsen, and Christina Lioma. 2016. Power Law Distributions in Information Retrieval. *ACM Transactions on Information Systems* 34, 2 (2016), 8:1–8:37.

[40] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (Cascais, Portugal) *(SOSP '11)*. 85–100.

[41] David M. W. Powers. 1998. Applications and Explanations of Zipf's Law. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning* (Sydney, Australia) *(NeMLaP3/CoNLL '98)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 151–160. http://dl.acm.org/citation.cfm?id=1603899.1603924

[42] Pierangela Samarati and Latanya Sweeney. 1998. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Harvard Data Privacy Lab.

[43] Kunwadee Sripanidkulchai. 2001. The Popularity of Gnutella Queries and its Implication on Scalability. http://www-2.cs.cmu.edu/kunwadee/research/p2p/gnutella.html.

[44] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2013. Path ORAM: An Extremely Simple Oblivious RAM Protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*. 299–310.

[45] John Walker. 2006. Mathematics: Zipf's Law and the AOL Query Database. https://www.fourmilab.ch/fourmilog/archives/2006-08/000740.html.

[46] Jiafan Wang and Sherman S. M. Chow. 2019. Forward and Backward-Secure Range-Searchable Symmetric Encryption. *IACR Cryptol. ePrint Arch.* 2019 (2019), 497. https://eprint.iacr.org/2019/497

[47] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2016. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *25th USENIX Security Symposium (USENIX Security)*. USENIX Association, Austin, TX, 707–720. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/zhang