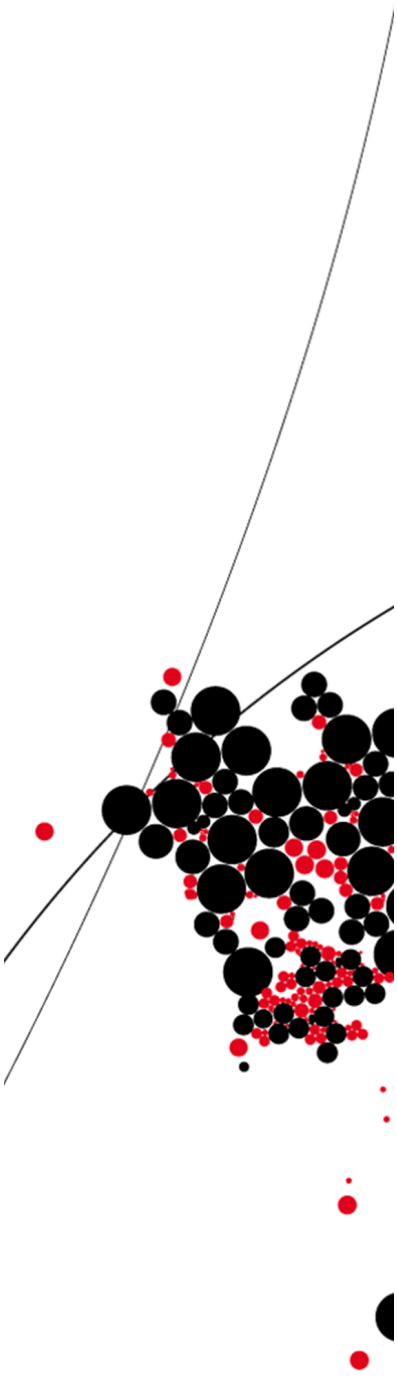
UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**

Defending against Access Pattern Attacks on Secure Range Query Schemes

Jasper J. F. Boot

Master Thesis
October 2020



Supervisors:
dr. A. Peter
dr. ing. F.W. Hahn
R.F. de Vries (KPMG)

Services and Cyber-Security Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

***Abstract*—We investigate a defense for secure range query schemes against existing attacks. Secure range query schemes make it possible to encrypt data while still being able to execute range queries on the data. The goal is that both the data and the queries are not leaked to an adversary. However, attacks exist on secure range query schemes which require only the access pattern leakage in order to reveal plaintexts. There is no known method to remove access pattern leakage. To make these schemes more secure, we obfuscate the access pattern by adding false positives. We tested this against state-of-the-art attacks to show the effectiveness of the measure. Since securing these schemes is inherently a trade-off between performance (in terms of communication overhead in this example), security (in terms of success rate of existing attack) and functionality (in terms of accuracy of the returned results), the experiments focus on these aspects to provide insight in the trade-offs between these factors. We provide a novel protocol (Local Indexed Search - LIS) using a local index which costs performance in order to make sure the only leakage is indeed the access pattern leakage. The results are open to interpretation; it takes at maximum 3.30 times the communication overhead to get at an acceptable security level (between 0.4 and 0.5 maximum symmetric error). It depends on the application context what the right trade-off is between security and performance.**

***Index Terms*—Searchable Encryption, Access Pattern, Range Queries, Defense**

I. Introduction

Traditional encryption encrypts a plaintext into something that is unreadable for someone without having the right key to decrypt it. In general, encrypted data loses certain properties making it impossible to search through this data without first decrypting it. This is problematic when it is crucial to search through the data before it can be used. We define an untrusted party that provides data storage as a service and call it the **service provider**. A service provider could be a cloud provider which offers data storage and backups. We consider the following scenario.

Scenario

When medical records are stored with a service provider, a doctor might want to search for patient information using an age range. But the doctor does not want to reveal the patient files, information about the age of patients or the entered search terms to the service provider. The doctor creates a search query with a lower bound and an upper bound forming a range. For example, patients with an age of 20 (a) or higher and and age of 30 (b) and lower.

In this scenario it would make no sense storing the decryption key with the cloud provider; the cloud provider (or someone with access to the system of the cloud provider) can then decrypt and read all of the data. The decryption key should reside with the user, meaning that the data will be unreadable for the service provider, making it impossible to search through it. In order for the scenario to work, the user needs to download all files from the service provider, decrypt them, and then search for the right file. Doing so directly omits one of the main uses of this system: not having to download all patient files on the device of the user.

Searchable encryption is an active field of research that investigates methods of storing data encrypted - so an adversary cannot read the stored data - while maintaining the ability to search through this data. There is no perfect way of accomplishing this yet. Some methods are slow but rather secure such as ORE [1] [2] and OPE [3]. Faster methods offer in turn less security [4]. It is crucial to find the right balance between security (in the sense of how much resources an attacker would require in order to get useful information out of the encrypted data), performance (in the sense of how much overhead it takes in communication and execution of the protocol) and functionality (in the sense of the accuracy of the data that is returned to a user). The trade-off is depicted as a triangle as shown in Figure 1.

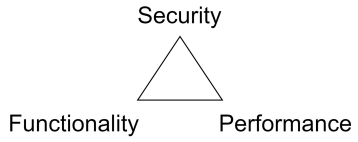


Fig. 1: The three main concerns when developing a searchable encryption scheme.

We focus on searchable encrypted range query schemes - schemes where a search query consists of two values (a and b) which act as an lower and upper bound for the search. For example, every patient with an age between 20 (a) and 30 (b). Everything within and including these values will be returned as the answer for the query. To accomplish this, the service provider requires extra information about the ordering of the data (although there are exceptions where it is not specifically required but these are generally slow). Leakage can occur both in the queries (because the service provider needs to be able to compare if results fall within the range) as well as in the results (as all elements in a result are neighbours in the dataset). Researchers investigated these leakages and crafted attacks with the goal of recovering e.g. the order of the encrypted data or even recovering the plaintexts of the encrypted data.

An example of the complete scenario is depicted in Figure 2.

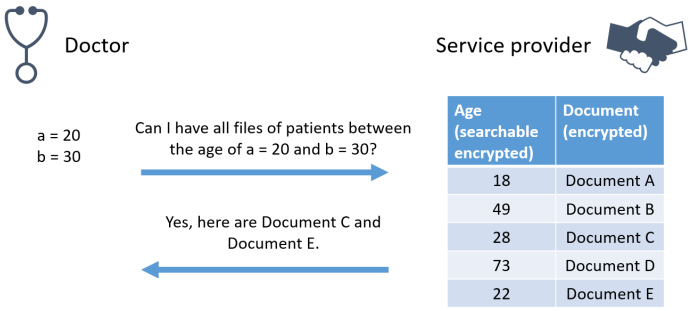


Fig. 2: Example of the secure range query scheme as stated in the Scenario.

We investigate searchable encrypted range query schemes which would only have an access pattern leakage in order to defend against them. We introduce a novel protocol (Local Indexed Search - LIS) which has this property. To defend against

access pattern attacks, a defense mechanism is introduced: adding false positives to the results of a query (and thus to the access pattern). The stated scenario would then not only return values within the range a - b, but also values outside the range. The goal is to make it harder for attacks which rely on constraints or on statistics to succeed. We investigated how much influence this has on existing attacks to provide insight in the trade-offs between security, functionality and performance. We used the amount of added false positives as a parameter. The goal is to give insight in a practical example of defending against attacks and furthering the field of research in the process. Creating this defense could potentially lead to the creation of better working attacks, which in turn should lead to the creation of better searchable encryption schemes in the categories security, functionality and performance.

II. Definitions

Most searchable encryption schemes consist of two different parts. A piece of data is encrypted with traditional encryption - which could be the patient file from the scenario. Such a file is called a document. The age is then stored separately using searchable encryption. When searching for a specific age range, not only the ages are returned but the documents as well. Different kinds of encryption can be combined to enhance the utility of the scenario. Figure 2 shows this structure as well.

In order to be able to discuss leakage it is important to define what kind of scheme we try to defend. It has the following properties:

1) **Range query scheme** We focus solely on range query schemes because this search type has not as extensively been researched as for example exact keyword match schemes.
2) **Access pattern leakage** The most basic type of leakage is access pattern leakage, defined as an attacker intercepting encrypted responses belonging to a query. The attacker can identify the specific documents being accessed when a query is executed. This leakage seems to be the hardest to avoid and more modern attacks only focus on abusing the access pattern.

The above mentioned properties can be found in every known range query scheme. The problem is that these schemes allow for more types of leakage than just our properties. To limit the leakage and make the setup for the experiments less complex we introduce our own simple scheme. It also makes it possible to introduce false positives in the experimental setup. We call this scheme **L**ocal **I**ndexed **S**earch - LIS for short. The idea is that a user encrypts documents with traditional encryption and stores them with a unique identifier at the service provider. The user then creates an index of search terms on their local device instead of at the service provider. A user can execute a search query - for example all patients with an age between 20 and 30, as in the scenario - and look up the corresponding document identifiers using their local index. The user requests these documents from the service provider. A depiction of the protocol can be found in Figure 3 and a description of the steps is given below.

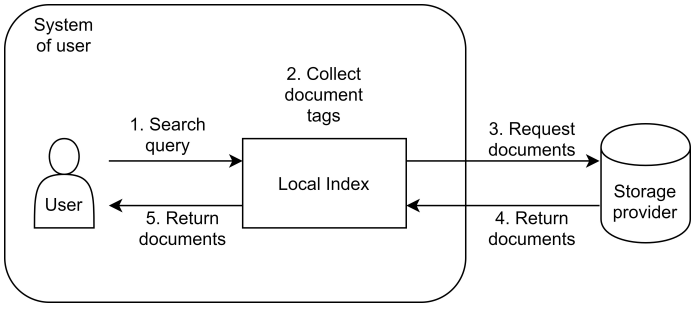


Fig. 3: Depiction of the LIS protocol.

1) The user enters a search query using the variables a and b. To search for everything between and including 20 and 30, the user would enter $a = 20$ and $b = 30$.
2) The Local Index searches which documents fall within the provided query and collects the identifiers of the corresponding documents.
3) The documents are requested at the service provider by using the document identifiers without revealing the search query.
4) The requested documents are returned to the user.
5) The requested documents are returned to the user. This is the same step as step 4, but this will change when we introduce the false positives.

LIS can now be extended to support our defense mechanism: adding false positives to the access pattern. Since the user stores a local index, it is possible to request more data than strictly required to complete the search query. By adding more document identifiers in step 2 which are outside the queried range to the list of documents to be requested from the storage provider, we obfuscate the access pattern. We will refer to this from now on with **adding false positives**. Since the false positives are added at the system of the user, it is possible to filter them out when the results are received. Step 4 and step 5 in the diagram show this. In step 5 the user receives only the requested results, while step 4 also contains false positives. The access pattern we investigate is basically the communication in step 4 of this diagram. Figure 4 shows the position of the adversary and where false positives are added.

III. Attack model

This section describes the capabilities we consider an adversary has. We consider the scenario in which the main goal of the adversary is finding the plaintexts of a dataset. In the scenario presented in our introduction (and depicted in Figure 2) this means finding out what the ages are which are stored at the service provider. We also make the following assumptions on the adversary:

- **Full control over service provider** We consider either that the adversary is the service provider or that the adversary has full control over the service provider. The

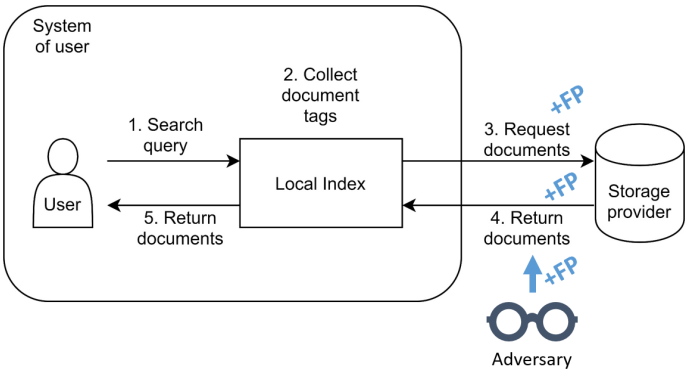


Fig. 4: Depiction of the LIS protocol including the position of the adversary and the false positives.

latter could be the case when the servers of the service provider are for e.g. hacked by the adversary.

- **Passive attacks** We only consider passive attacks, which means that the information the adversary sees can only be read and not altered in any way to alter the execution of the protocol.
- **Plaintext domain is known** We assume that the domain of the plaintext data is known to the adversary. The domain is required for the attacks to work. For example, in the scenario using ages can even be easily deducted.
- **Query distribution is known** We assume that the query distribution is known to the adversary. Although this is a very powerful assumption, it seems unavoidable in existing attacks.

These assumptions can be summarised as an honest-but-curious service provider as our adversary.

As stated earlier, we focus on the access pattern leakage specifically because this is generally not possible to get rid of - besides using obfuscation as we do. The access pattern consists of the individual memory locations accessed on the server when a query is being executed. The memory locations can be identified, but the data in the memory locations are encrypted records. Simplified, the unique identifiers of the datapoints stored on the server. The adversary thus sees sets of unique identifiers belonging to a query but not the query itself or the actual data values. To map it to the LIS protocol, this is the same as step 4 in the protocol, which is the set of encrypted documents. We thus assume that the adversary can only intercept the data transferred in step 4.

IV. Attacks on SE range query schemes

There exist many attacks on searchable encrypted range query schemes as shown in Appendix section A. It shows per attack what it requires to succeed, what constraints it has and what the goal of the attack is. The overview also contains references to relevant papers describing the attacks.

To select attacks applicable to our research, requirements have been defined. With these requirements a selection of attacks has been made from the overview in Appendix section A. The requirements are focused on the attacker having minimal knowledge of the plaintext data, attacks using access pattern leakage and considering the established attack model. Minimal knowledge means that we restrict the requirements as much as possible without excluding all attacks. The requirements follow below.

- **The goal is Plaintext Recovery** We focus on attacks which aim to recover plaintexts, excluding e.g. attacks focusing on recovering order or query plaintexts.
- **Targets access pattern leakage only** We only focus on access pattern as this type of leakage seems unavoidable. Communication volume can be directly obtained from the access pattern, as this is the amount of documents returned by a query.
- **No plaintexts are required** Since we want to make minimal assumptions on the side of the attacker, we omit attacks which specifically depend on knowledge about plaintexts of the data or plaintext queries in order to succeed.
- **No plaintext distribution is known** Since we want to make minimal assumptions on the side of the attacker, we omit attacks which specifically depend on knowledge about the distribution of plaintext in order to succeed. This is also referred to as auxiliary information.
- **Dense dataset is not required** We focus on attacks which do not depend on a dense dataset to succeed. In a dense dataset every domain value occurs at least once.

As stated in the previous section, we do assume the adversary has knowledge over both the plaintext domain and the query distribution as this seems unavoidable.

When these constraints are applied to the mentioned attack overview from Appendix section A the following attacks are of our interest:

1) Access Pattern Attack [5]
2) Communication Volume Attack [5]
3) e-Approximate Database Reconstruction Attack [6]

The work of Grubbs et al. [6] directly references the work of Kellaris et al. [5] by enhancing the methods used. Grubbs et al. created a generalized version of the attacks of Kellaris et al. which is called the GeneralizedKKNO Attack. This attack directly builds onto the attacks in the paper of Grubbs et al. [6] generalizing them to be e-Approximate. e-Approximate means that the algorithm always succeeds and that the attacker can calculate how much error there is in the plaintexts uncovered by the algorithm. The e-Approximate Database Reconstruction Attack (a.k.a. the ApproxValue Attack) [6] continues this path with a more optimized algorithm. Because the same methods are used we only focus on the newest form: the e-Approximate Database Reconstruction Attack [6].

V. The ApproxValue Attack

A. Overview

The ApproxValue attack is created by Grubbs et al. [6] and builds directly upon the Access Pattern Attack as described in the work of Kellaris et al. [5]. The adversary has access to the access pattern as shown in Figure 4 and tries to estiamate the

real values (ages for example, according to the scenario). The main principle of these attacks is that they rely on statistics. The amount of queries a specific record should appear in out of all possible queries on the domain is used to estimate its value. It is then assumed that the distribution of how queries are generated is fixed and known; in this case queries are assumed to be generated uniformly random. Hence, datapoints at the beginning or the end of the domain are queried less often than the datapoints in the middle of the domain. This distribution is depicted in Figure 5. The attack measures the amount of queries a specific value is in from all intercepted queries. Compared with the theoretical distribution it provides an estimate of its real value. The interesting aspect of the ApproxValue attack is that it always succeeds, but precision of the results is required to interpret the results. The results will be more precise when more queries / access patterns are intercepted. It is possible to calculate - using the amount of used queries - the preciseness of the outcome, giving an error margin in which the predicted values will reside. In practice, to obtain meaningful results, an adversary potentially requires to intercept less queries. The age of a person is a good example: It might not be possible to predict the real age values, but for example within a 10 year error margin. Depending on the goal of the adversary, it can still be useful.

B. Algorithm

The algorithm of the ApproxValue attack is included in algorithm 1 as described by Grubbs et al. [6]. To explain how the attack works we will use both the algorithm as well as some graphs. These graphs will contain ideal scenario's where real-world measurements will always have an error margin.

Algorithm 1: ADR Algorithm ApproxValue created by Grubbs et al. [6]

Input : Set of queries $\mathcal{Q}$.
Output: Function *est-val* approximating val.

1 **for** *each record* r **do**
2 $\quad c(r) \leftarrow |\{q \in \mathcal{Q} : r \in q\}|/|\mathcal{Q}|$
3 $\quad \widetilde{v}(r) \leftarrow \arg\min_k |c(r) - p(k)|$
4 **end**
5 $r_A \leftarrow \arg\min_r |\widetilde{v}(r) - N/4|$
6 $\widetilde{v_A} \leftarrow \widetilde{v}(r_A)$
7 **for** *each record* r **do**
8 $\quad c'(r) \leftarrow |\{q \in \mathcal{Q} : r_A, r \in q\}|/|\mathcal{Q}|$
9 $\quad \widetilde{w_L} \leftarrow \arg\min_{k \in [1, \widetilde{v_A}]} |d(\widetilde{v_A}, k) - c'(r)|$
10 $\quad \widetilde{w_R} \leftarrow \arg\min_{k \in [\widetilde{v_A}, N]} |d(\widetilde{v_A}, k) - c'(r)|$
11 $\quad$ **if** $c(r) < (p(\widetilde{w_L}) + p(\widetilde{w_R}))/2$ **then**
12 $\quad\quad est\text{-}val(r) \leftarrow \widetilde{w_L}$
13 $\quad$ **else**
14 $\quad\quad est\text{-}val(r) \leftarrow \widetilde{w_R}$
15 $\quad$ **end**
16 **end**

First we introduce $p(k) = Pr(A_k) = \frac{2}{N(N+1)}(N+1-k)$ - the probability that a uniformly random range contains the value k. A graph of this distribution is depicted in Figure 5.

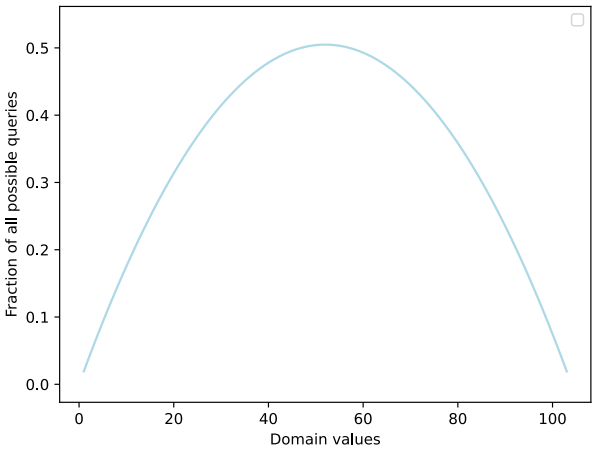


Fig. 5: ApproxValue algorithm step 1. The blue line depicts the distribution between domain values and the fraction of all possible queries this domain value appears in.

In line $1-3$ of the algorithm two distributions are made to be used later. Distributions are made for every record found in the set of queries $\mathcal{Q}$. On line 2 the amount of intercepted queries a record is in, is divided by the total amount of intercepted queries generating a fraction of the measurement. On line 3 this is then mapped to distribution $p(k)$ to obtain an approximate value for this record. Note that the distribution is symmetric, meaning that there are (in general) two points in the distribution which match. Only the leftmost point will be returned.

On line 5 the anchor record is chosen. The anchor record is as close to the first quarter of the domain as possible. The record corresponding to the estimated value closest to $N/4$ is chosen. In line 6 the approximated value of this record is also stored according to the distribution of $p(k)$.

Lines $7-16$ contain the main loop which iterates over all records and returning the approximated value. One of the main challenges here is that our earlier estimate only gives an estimate up until reflection because the distribution of $p(k)$ is symmetric.

We now introduce $d(v_A, k)$ as follows:

$$d(v_A, k) \stackrel{def}{=} \frac{2}{N(N+1)} \cdot \begin{cases} k(N+1-v_A) & \text{if } k \leq v_A \\ v_A(N+1-k) & \text{if } k > v_A. \end{cases}$$

$d(v_A, k)$ shows the fraction of all queries both v_A and k are in, which will be used in the next section.

On line 8 $d(v_A, k)$ is used to combine every record (as k) with the anchor point v_A. The idea is to use the anchor point and calculate a fraction of the records to get an estimate of its value - like line 2 does. The difference is that we check the amount of intercepted queries containing both the current record as well as the anchor point, resulting in a different distribution. A depiction of this distribution is in Figure 6

as the dotted blue line where the (ideal) anchor point is the highest point.

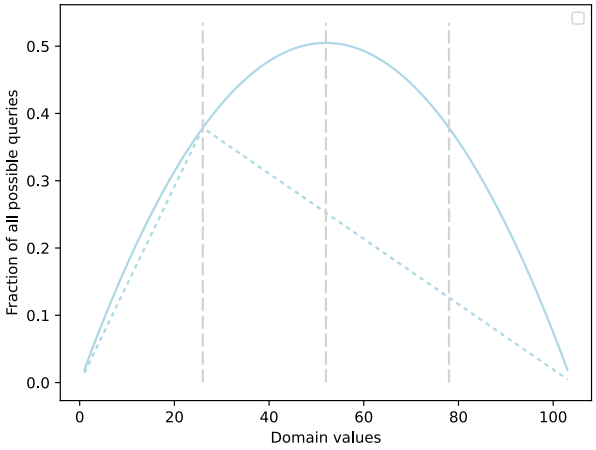


Fig. 6: ApproxValue algorithm step 2. The blue dotted line depicts for every domain value in what fraction of all possible queries both this domain value and the value on a quarter of the domain appear in. The grey dotted lines represent a quarter, half and three quarters of the domain values respectively.

In Figure 7 two points are added. The blue star is the real record we try to find the value of plotted on the blue line. The blue dot is not the real data value but has the same value on the blue line. This means that by just using the $P(k)$ distribution, these points are indistinguishable. Figure 8 shows a orange dotted line which is on the record we investigate. An orange star has been placed along that line on the yellow line. In this case this point will be the value chosen by line 9.

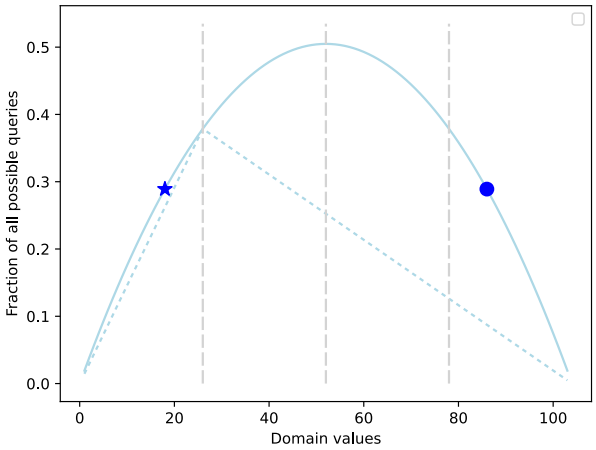


Fig. 7: ApproxValue algorithm step 3. The blue star marks the currently investigated domain value mapped on the blue line, the blue dot maps the same fraction on the blue line as the blue star.

In lines 9 and 10 a left point and a right point are chosen - depicted in Figure 9. The real record value has a orange dotted

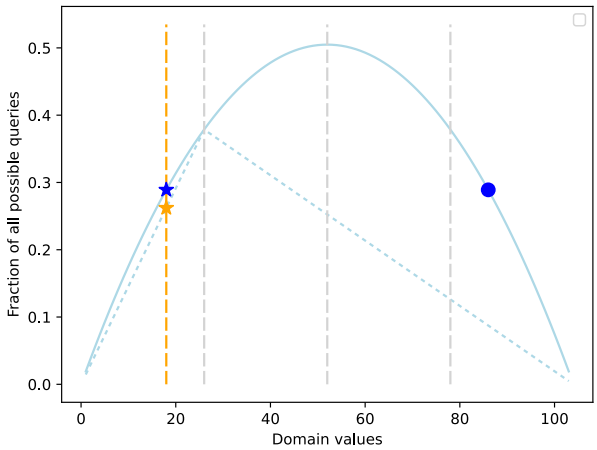


Fig. 8: ApproxValue algorithm step 4. The orange dotted line and the orange star are placed on the same domain value as the blue star. The orange star is mapped on the blue dotted.

line through it and an orange star where it crosses the blue dotted line. The other orange dotted line and point are on the same value on the blue dotted line. There is also a blue dot mapped on the blue line on the same value. These two values on the blue dotted line are indistinguishable. The problem here is that we need to choose between the left and the right value, as only one is the correct one as shown in the figure.

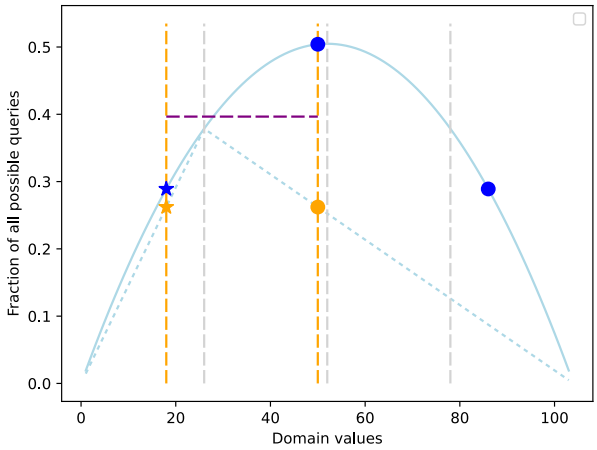


Fig. 9: ApproxValue algorithm step 5. The new orange dotted line is placed on the same fraction on the blue dotted line as the orange star. The orange dot and the blue dot are mapped onto the blue dotted line and the blue line along the orange dotted line.

On line 11 this distinction is made, which is depicted in Figure 9 as the horizontal dashed purple line. It takes the average of both the left and right point on the $p(k)$ distribution (blue line). If the measured value from the blue line (the blue star) is lower it picks the left point, otherwise it picks the right point. In lines 12 and 14 these values are returned.

C. Results as reported by Grubbs et al. [6]

The experimental results (Figure 10) are directly borrowed from the paper of Grubbs et al. [6] and are used to understand the attack and to evaluate our implementation of it. We focus on the topmost colored lines. The horizontal axis shows the amount of queries captured by the attack and the different lines mean different domain sizes. They use 1,000 datapoints in the database. On the vertical axis they used a specific metric to evaluate how well the attack preformed - the maximum symmetric error as a fraction of the domain size.

The maximum symmetric error is defined by taking the maximum of the following formula:

$$|min\{\text{est-val}(r), N + 1 - \text{est-val}(r)\} - \text{symval}(r)|$$

Where est-val(r) is the result of the ApproxValue algorithm and symval(r) is defined as:

$$\text{symval}(r) \stackrel{def}{=} min\{val(r), N + 1 - val(r)\}$$

The maximum symmetric error is a measurement to compensate for the fact that an estimated value can be both on the left half of the values or on the right half. The measurement negates this difference. The maximum part means that we are only concerned in the maximum error because an attacker does not know how precise the results are up until the maximum possible error. In the paper it is explained how this upper-bound can be calculated as a fraction of the domain size using the number of intercepted queries. It is also shown in Figure 10 as the grey dotted line. They show that their experimental results indeed stay below this line.

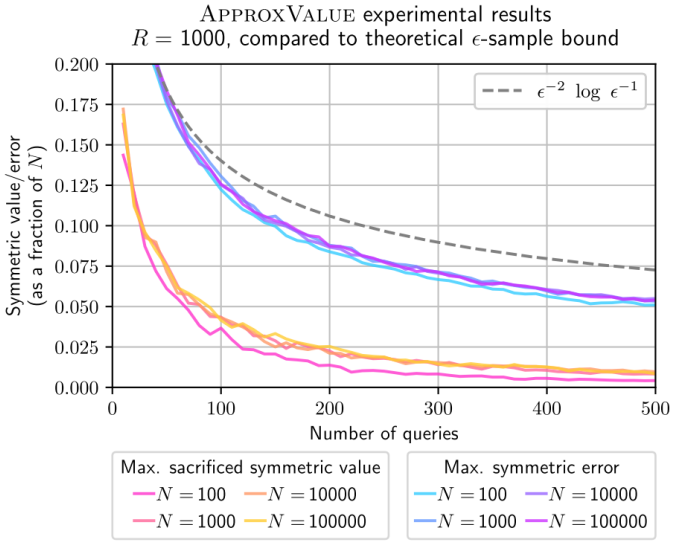


Fig. 10: Experimental results of the ApproxValue attack as presented in the paper by Grubbs et al. [6]

D. Experimental setup

In our experimental setup we used the same variables and assumptions as the researchers did with the ApproxValue attack. We created an implementation as there was no implementation of the attack available. We verified the results with the results from the paper (Figure 10) by creating the same graph. We expect that the results are the same if the implementation of the attack is correct. The result of this verification is shown in Figure 11. One key difference is that we did not measure with 500 different databases, but only with 5. This graph gives us confidence the results of the paper are indeed correct and we implemented the attack well.

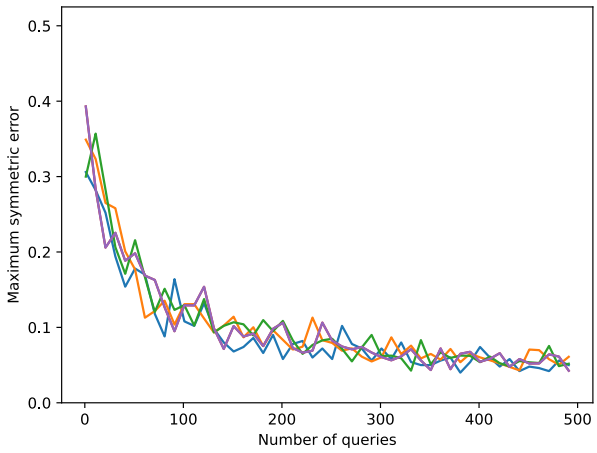


Fig. 11: Graph showing the results of the implemented ApproxValue attack without any defense mechanisms. It depicts the amount of queries on the horizontal axis and the masimum symmetric error as a fraction of the domain size on the vertical axis. The colors represent different domain sizes; blue=100, orange=1,000, green=10,000, purple=100,000

E. Intuition

Because the ApproxValue attack relies heavily on the two distributions $p(k)$ and $d(v_A, k)$ and do not take wrong measurements into account our hypothesis is that when adding false positives to the queries the results of the attack will drastically drop. When looking at Figure 11 the line should ideally stay high instead of going down to indicate that the attack results are significantly worse.

VI. Methodology

To test the impact of false positives on the success of attacks we create an experimental setup. Within this setup we create an abstract representation of the attack, defense and protocol to conduct measurements to the success of the attack in relation to the amount of false positives added.

A. Experimental setup

With the experiment we want to see how well the proposed defense - adding false positives to query results - works on

the previously discussed topics; security, performance and functionality. A setup is implemented in code which generates random datasets, random queries on these datasets and executes the ApproxValue attack on them. It collects the results in terms of Maximum symmetric error as a fraction of N. For simplicity, we only use numeric values as datapoints. We can now set parameters for the setup and run the experiment. The result will be captured in a graph.

The setup contains different parameters which are listed below.

Domain A fixed domain is used to when a dataset is created. The domain is also known to the adversary. The datapoints within the dataset will all be within the domain.

Datapoints The amount of datapoints which will make up the dataset. It could be that a domain value has multiple datapoints or that a domain value has no datapoint.

Query amount The amount of queries generated (and thus captured by the adversary). The queries are generated uniformly at random within the set domain.

Run amount The amount of times a measurement will be repeated. This is used to smooth out the results as there is randomness involved in generating datasets and queries. The results of the different runs will be averaged.

Adversary The adversary which will try to identify the real datavalues - either the RandomAdversary or the ApproxValueAdversary. The RandomAdversary is a naive implementation which returns a random value from the domain for every result, used to create a baseline for the experiments. The ApproxValueAdversary is the implementation of the ApproxValue algorithm as explained before.

False positives The fixed amount of false positives added to the query results sent to the adversary. The false positives could also be zero. They are picked at random from all datapoints which are not in the given query. When there are no available datapoints left, it could theoretically be that less false positives are added.

In order to provide insight in how well the false positive defense works and what the trade-offs in terms of security, functionality and performance are we generate different graphs.

The security is measured in terms of how well the attacks succeed in recovering the plaintexts. This is achieved by implementing the attacks and measuring the results when facing randomly generated datasets and queries. Two baselines are generated: one with the ApproxValue adversary and no false positives and one with the RandomAdversary and no false positives. These baselines give the boundaries for the experimental results which will fall somewhere between those - they enable us to discuss the results and put them into perspective.

The performance is measured in terms of communication overhead. The absolute value will match the amount of false positives injected. The relative value will match the following formula:

$$\frac{\text{query result size} + \text{false positives}}{\text{query result size}}$$

It is possible however to have a denominator of zero, meaning that it is not possible to calculate the formula and have an infinitely large overhead. As we do not want to omit these results, we change a zero denominator to the value 1.

The functionality is not measured because the precision of the query results does not change when the defense is applied. False positives are added by the client which can be thrown away again automatically when received. This means that the returned results always exactly match the query.

The results will be in the form of two graph types:

Graph type A Depicts on the horizontal axis the amount of added false positives and on the vertical axis the maximum symmetric error as a fraction of the domain size. When the amount of added false positives is zero the attack shows results as described in the attack paper.

Graph type B Depicts on the horizontal axis the amount of queries the attack used and on the vertical axis the maximum symmetric error. When the amount of added false positives is zero the attack shows results as described in the attack paper. In general, the symmetric error should lower when the amount of queries increase. When false positives are added we expect the line to go down less. The higher the graph, the worse the attack preforms.

We are also interested in two specific variables: The domain size and the amount of generated queries. We plot these as different lines on the same graph - graph type A. This should give an idea if these variables matter for the performance of the attack, and if it does, what this impact is.

VII. Results

A. Random baseline

A random baseline is generated as described in the previous section, which is used to show what a random adversary looks like to compare it to the results of the ApproxValue adversary later. The used settings are depicted in Table 1. The graph with the results is depicted in Figure 12.

Domain	1 - 100
Datapoints	1,000
Query amount	500
Run amount	5
Adversary	RandomAdversary
False positives	Varied between 0 and 200 in increments of 10.

Tab. 1: Properties of the random baseline experiment.

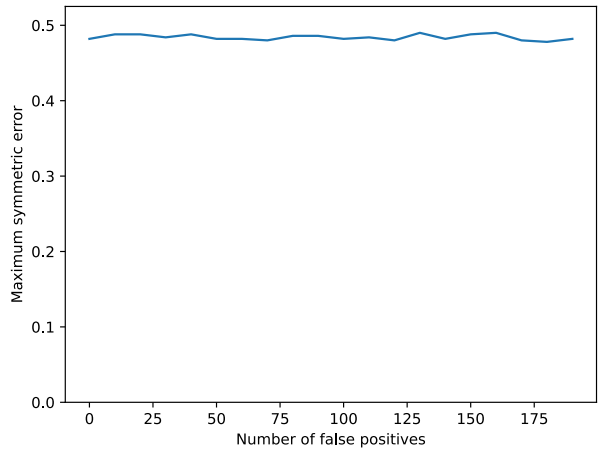


Fig. 12: Graph type A for the random adversary, depicting the amount of false positives added on the horizontal axis and the maximum symmetric error as a fraction of the domain size on the vertical axis.

B. Graph type A with varying domain sizes

The ApproxValue attack in the form of graph type A with varying domain sizes. On the horizontal axis are the absolute number of added false positives. On the vertical axis is the maximum symmetric error. The lower the graph, the more successful the attack is. The graph is generated with the settings depicted in Table 2. The results can be found in Figure 13.

Domain	1 - [100, 1,000, 10,000, 100,000]
Datapoints	1,000
Query amount	500
Run amount	5
Adversary	ADRAdversary
False positives	Varied between 0 and 200 in increments of 10.

Tab. 2: Properties of the ApproxValue graph type A with varying domain sizes experiment.

C. Graph type A with varying datapoints

The ApproxValue attack in the form of graph type A with varying datapoints. On the horizontal axis are the absolute number of added false positives. On the vertical axis is the maximum symmetric error. The lower the graph, the more successful the attack is. The graph is generated with the settings depicted in Table 3. The results can be found in Figure 14.

D. Graph type B

The ApproxValue attack in the form of graph type B. On the horizontal axis are the amount of intercepted queries / access patterns. On the vertical axis is the maximum symmetric error. The lower the graph, the more successful the attack is. The

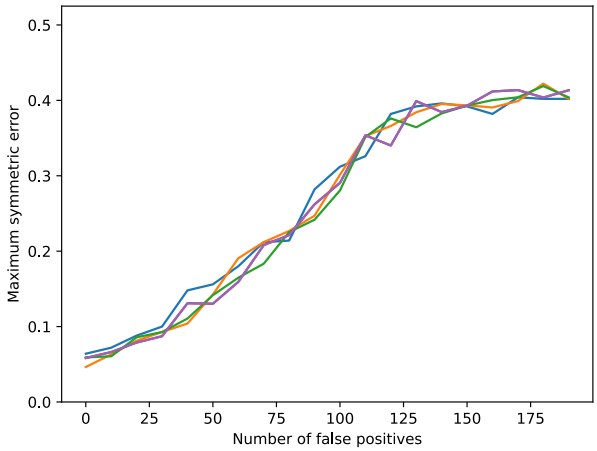


Fig. 13: Graph type A for the e-Approximate Database Reconstruction attack, depicting the amount of false positives added on the horizontal axis and the maximum symmetric error as a fraction of the domain size on the vertical axis. The colors represent different upper bounds for the domain; blue=100, orange=1,000, green=10,000, purple=100,000.

Domain	1 - 100
Datapoints	[10, 100, 1,000, 10,000]
Query amount	500
Run amount	5
Adversary	ADRAdversary
False positives	Varied between 0 and 200 in increments of 10.

Tab. 3: Properties of the ApproxValue graph type A with varying amounts of datapoints experiment.

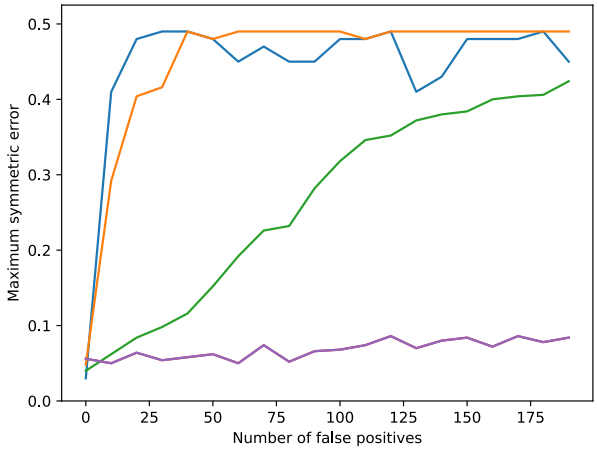


Fig. 14: Graph type A for the e-Approximate Database Reconstruction attack, depicting the amount of false positives added on the horizontal axis and the maximum symmetric error as a fraction of the domain size on the vertical axis. The colors represent different amounts of datapoints; blue=10, orange=100, green=1,000, purple=10,000.

attack also tends to be more successful when it intercepts more queries. The graph is generated with the settings depicted in Table 4. The results can be found in Figure 15.

Domain	1 - 100
Datapoints	1,000
Query amount	Varied between 1 and 501 in increments of 10.
Run amount	5
Adversary	ADRAdversary
False positives	[0, 50, 100, 150, 200, 250, 300]

Tab. 4: Properties of the ApproxValue graph type B.

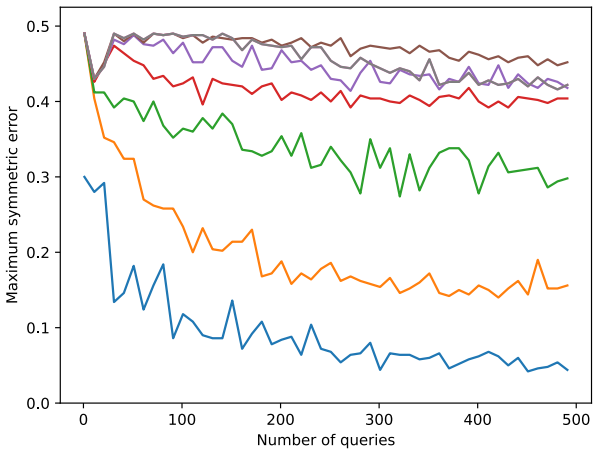


Fig. 15: Graph type B for the e-Approximate Database Reconstruction attack, depicting the amount of queries on the horizontal axis and the maximum symmetric error as a fraction of the domain size on the vertical axis. The colors represent different amounts of injected false positives; blue=0, orange=50, green=100, red=150, purple=200, brown=250, grey=300.

VIII. Discussion

The results give different perspectives on the problem and will be discussed per graph. Afterwards, the results as a whole will be discussed as well as what they mean in practice.

The random baseline is generated in the form of graph type A (Figure 12). It shows that the false positives do not have a clear impact on the results; which is as expected. We can see that between a maximum symmetric error of 0.4 and 0.5 true randomness is approached, giving a clear idea when the attack does not work. If the results of an attack are the same as the random baseline, they provide no meaningful insight to the adversary. We also know from Figure 10 and Figure 11 that with the query amount parameter set to 500 the expected results from the attack should be around 0.05, meaning that these are the maximum values we expect in the other results. There is a gray area however; an attack does not either work or does not work. It can depend on the context of a dataset or the attack goals what is considered 'secure', but it definitely includes the random baseline. Lower values could also be 'secure' depending on the context. We will assume everything with a score of 0.4 and above is relatively secure.

Graph type A has been generated in two forms (one with varying domain sizes and one with varying amounts of datapoints in the database). The one with varying domain sizes (Figure 13) gives clear overlapping lines suggesting heavily that this parameter does not influence the outcome of the experiment. Furthermore it is evident that the false positives decrease the success of the attack. The scale however is equally important. The attacks reach the maximum symmetric error of 0.4 after approximately 130 false positives. Relative to the amount of datapoints in the database (1,000) this is intuitively quite a lot as it is added to all query results.

The other variant of graph type A - the one with varying amounts of datapoints (Figure 14) - pants a different image. The amount of datapoints influences the success of the attack clearly; if the amount of datapoints get higher, the success of the attack increases. We want to see how much false positives are needed for the attack to fail. We pick the first point for every graph where it passes the maximum symmetric error of 0.4. Please note that the step size of the false positives is 10. This gives us the results as depicted in Table 5. Results are a fraction of the average query result size of the experiment.

Amount of datapoints	Amount of false positives needed
10	3.30
100	2.52
1,000	2.27
10,000	N/A

Tab. 5: Amount of false positives needed for the maximum symmetric error to surpass 0.4. The data is taken from the graph in Figure 14. Please note that this graph has a step size of 10. It is calculated using the following formula:

$$\frac{\text{average query result size} + \text{false positives added}}{\text{average query result size}}$$

The impact of these numbers highly depends on the context of the application. Intuitively, 3.30 is quite high as it generates more than three times the amount of traffic. High amounts of overhead could be a problem for currently deployed databases. On the other hand, if the overhead is less of an issue than the security risks and the functionality increase, it is a good result. Data shows that the overhead tends to be lower when the amount of datapoints increase. Unfortunately, our implementation was not able to generate much larger cases as it would take too long to render. If the amount would go down further it would be good as this would lower the overhead in larger datasets.

Graph type B (Figure 15) shows similar results as graph type A. We can directly compare it to the results of the paper introducing ApproxValue [6] as shown in Figure 10 and Figure 11. As the blue line corresponds to no false positives, which is the same as the results from the paper. As expected,

the graph shows that the maximum symmetric error goes less down when the amount of false positives is increased.

It is evident that the method of adding false positives works but the real question is how well it works. It all comes down to the triangle between security, functionality and performance (Figure 1). Again, we will not discuss functionally here as this will not change with or without the proposed defense. However, both the security and the performance do change. The generated graphs show clearly what their trade-off is as they directly show this correlation. It depends on the situation if the defense is feasible and if yes, how much false positives should be added. There is also much to improve about the defense, which will be elaborated more on in section X.

IX. Related work

General solutions to defending access pattern leakage exist. The main method to achieve this is Oblivious RAM - ORAM [7] for short. Although it is difficult to directly compare their results to our scenario, these methods can be applied to it. ORAM creates quite some overhead, limiting the overall performance. It starts with returning all documents to a search query - infeasible for our scenario. But there are works which greatly improve upon this. Goldreich et al. [7] mention a "poly-logarithmic slowdown in the running time". Garg et al. [8] investigates this problem as well ending up with a communication cost of $O(log(n)loglog(n))$ where n is the number of documents, specifically applied to exact keyword match searchable encryption. And Stefanov et al. [9] mention a $O(logN)$ overhead in specific cases. Our approach however does not require a different implementation at the server side or the use of multiple rounds.

Chen et al. [10] applied ORAM techniques to searchable encryption schemes, although with exact keyword match instead of range queries. They used differential privacy to back their claims. In their work, two interesting differences occur. Their security guarantees are higher than ours (using differential privacy) but their measured communication overhead is as well: they mention "2x" communication overhead up until "5x 6x" communication overhead [10] depending on the attack. Besides communication overhead, their documents also need to be saved redundantly, creating a potentially large storage overhead as well. However, comparisons with our approach can be difficult, as the setting is slightly different.

Different methods of range query schemes exist. Range query schemes could also be constructed using Order Preserving Encryption [3] or Order Revealing Encryption [1] [2]. However, OPE and ORE suffer from other kinds of leakages themselves. They can reveal e.g. the ordering of the plaintexts, which can (in)directly lead an adversary to retrieve the plaintexts as well.

X. Future work

The method as proposed in this paper could be improved in various ways which we elaborate on in this section.

We used only a fixed amount of false positives. Assessing the impact of varying the amount of false positives added per query could be vital to lowering the overhead. In our LIS protocol it is possible to know the amount of results which a query returns in advance, so the amount of false positives could be changed to a fixed overhead percentage per query. Different methods of selecting the false positives could be explored. We selected them at random. When selecting them to create a specific distribution could potentially lead to better results. Or select them further away from the real range query.

We did not go into detail about functionality as shown in the triangle in Figure 1. When false negatives are introduced this could drastically change, causing results which the user expects to not be returned, lowering the functionality. However, it could mean that false negatives have an influence on the attack success as well. Investigating the impact of these false negatives can prove interesting to specific use cases. For example it can lead to enhanced security but it can only be used in a context where a less than perfect functionality is acceptable.

Our test setup was limited to a single attack and attack goal. More attack goals can be investigated, like e.g. Order Reconstruction - which could (sometimes trivially) lead to Plaintext Reconstruction.

LIS could also be improved and extended. It would be interesting to see how well LIS would work in real-world scenarios. If this is the case, LIS could improve security quite a lot and bring searchable encryption closer to being used in practice.

Searchable encryption in general would benefit from more research after the trade-offs between security, functionality and performance. This trade-off is vital and could potentially take the field of research from the drawing board to real-world applications.

XI. Conclusion

Adding false positives to query results does significantly decrease attack success - meaning the amount of information an adversary can obtain is close to random (between 0.4 and 0.5 maximum symmetric error). However, it depends highly on the context of the application to assess how well this works. Adding false positives could even require a communication overhead of 3.30 but it does provide security against these attacks. It is also possible to scale down this security in order to decrease the performance overhead. It all comes down to the trade-offs between security, functionality and performance (Figure 1).

There are quite some extensions left to be investigated for the defense, which makes it promising. This means that 3.30 is an upper bound. It also depends highly on our created protocol (LIS) which only leaks the access pattern. The main difference is that we use a local index which increases security, while also increasing performance overhead. We also do not require a change in the protocols of the service provider.

Our approach is novel and the results mean it is feasible in practice, bringing secure range query schemes closer to be used in practice. Future research can strengthen this approach leading to even better results.

References

[1] K. Lewi and D. J. Wu, "Order-Revealing Encryption: New Constructions, Applications, and Lower Bounds," Tech. Rep., 2016.

[2] D. Bogatov, G. Kollios, and L. Reyzin, "A comparative evaluation of orderrevealing encryption schemes and secure rangequery protocols," in *Proceedings of the VLDB Endowment*, vol. 12, no. 8. PVLDB, 2018, pp. 933–947. [Online]. Available: https://doi.org/10.14778/3324301.3324309

[3] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6841 LNCS, 2011, pp. 578–595.

[4] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," p. 51, 2014. [Online]. Available: http://dx.doi.org/10.1145/2636328

[5] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 24-28-Octo, 2016, pp. 1329–1340. [Online]. Available: http://dx.doi.org/10.1145/2976749.2978386

[6] P. Grubbs, M. S. Lacharite, B. Minaud, and K. G. Paterson, "Learning to reconstruct: Statistical learning theory and encrypted database attacks," in *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2019-May, 2019, pp. 1067–1083.

[7] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[8] S. Garg, P. Mohassel, and C. Papamanthou, "TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9816, 2016, pp. 563–592.

[9] E. Stefanov, M. van Dijk, E. Shi, T.-h. Hubert Chan, C. Fletcher, L. Ren, X. Yu, S. Devadas, and U. Berkeley, "Path ORAM: An Extremely Simple Oblivious RAM Protocol," Tech. Rep. [Online]. Available: http://arxiv.org/abs/1202.5150v1

[10] G. Chen, T. H. Lai, M. K. Reiter, and Y. Zhang, "Differentially Private Access Patterns for Searchable Symmetric Encryption," *Proceedings - IEEE INFOCOM*, vol. 2018-April, pp. 810–818, 2018.

[11] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Inference attack against encrypted range queries on outsourced databases," in *CODASPY 2014 - Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*. Association for Computing Machinery, 2014, pp. 235–246.

[12] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 2015-Octob. Association for Computing Machinery, oct 2015, pp. 644–655.

[13] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart, "Leakage-Abuse Attacks against Order-Revealing Encryption," Tech. Rep.

[14] M. S. Lacharite, B. Minaud, and K. G. Paterson, "Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage," in *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2018-May. Institute of Electrical and Electronics Engineers Inc., jul 2018, pp. 297–314.

[15] P. Grubbs, M. S. Lacharité, B. Minaud, and K. G. Paterson, "Pump up the volume: Practical database reconstruction from volume leakage on range queries," in *Proceedings of the ACM Conference on Computer and Communications Security*. Association for Computing Machinery, oct 2018, pp. 315–331.

[16] E. A. Markatou and R. Tamassia, "Full Database Reconstruction with Access and Search Pattern Leakage," Tech. Rep.

[17] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution," Tech. Rep.

Appendix

Attacks table

A compact table with all the attacks is on the next page. It gives a quick overview of the attacks on searchable encrypted range queries. The table headers show properties an attack could require in order to be successful. Below is an explanation of the symbols.

Attack The name of the attack and a reference to the source.

Year The year the paper has been published.

Goal The goal of the attack. It could be one of the following: FPR = Full Plaintext Recovery
PPR = Partitioning of Plaintext Recovery
APR = Approximate Plaintext Recovery
AOR = Approximate Order Recovery
VCR = Value Counts Recovery
NRPR = New Record Plaintext Recovery
QPR = Query Plaintext Recovery

Access pattern If the attack uses access pattern leakage to succeed.

Search pattern If the attack uses search pattern leakage to succeed.

ORE/OPE If the attack uses leakage from Order Revealing Encryption of Order Preserving Encryption to succeed.

Communication volume If the attack uses communication volume leakage to succeed.

Query knowledge If the attack requires knowledge about queries to succeed. This could be query plaintexts.

Plaintext knowledge If the attack requires knowledge about plaintext data values to succeed.

Query distribution What assumptions the attack makes on the distribution of queries.

Plaintext distribution If the attack requires to know the plaintext distribution in order to succeed.

Dense data If the data requires to be dense for the attack to succeed.

The following markings are used.
✓= Required
●= Preferably
✗= Not required
* = Preferred, but not required

Attack	Year	Goal	Access pattern	Search pattern	ORE/OPE	Communication volume	Query knowledge	Plaintext knowledge	Query distribution	Plaintext distribution	Dense data
Inference Attack [11]	2014	FPR	✓	✗	✗	✗	✓	✗	Uniform*	✓	✗
Sorting Attack [12]	2015	FPR	✗	✗	✓	✗	✗	✗	N/A	✗	✓
Cumulative Attack [12]	2015	FPR	✗	✗	✓	✗	✗	✗	N/A	✓	✗
Access Pattern Attack [5]	2016	FPR	✓	✗	✗	✗	✗	✗	Uniform	✗	✗
Communication Volume Attack [5]	2016	FPR	✗	✗	✗	✓	✗	✗	Uniform	✗	✗
Non-crossing Attack [13]	2017	FPR	✓	✗	✓	✗	✗	✗	N/A	✓	✗
BCLO Attack [13]	2017	FPR	✓	✗	✓	✗	✗	✗	N/A	✗	✗
CLWW Attack [13]	2017	FPR	✓	✗	✓	✗	✗	✗	N/A	✗	✗
Partitioning Attack [13]	2017	PPR	✗	✗	✓	✓	✗	✓	N/A	✓	✗
Binomial Attack [13]	2017	FPR	✗	✗	✓	✗	✗	✗	N/A	✓	✗
Improved Reconstruction Attack [14]	2018	FPR	✓	✗	●	✗	✗	✗	Uniform	✗	✓
Improved Approximate Reconstruction Attack [14]	2018	APR	✓	✗	✗	✗	✗	✗	Uniform	✗	✓
Improved Auxiliary Distribution Attack [14]	2018	FPR	✓	✗	✓	✗	✗	✗	Uniform	✓	✗
Communication Volume Improved Attack [15]	2018	VCR	✗	✗	✗	✓	✗	✗	Uniform*	✗	●
Update Recovery Attack [15]	2018	NRPR	✗	✗	✗	✓	✗	✗	Uniform*	✗	●
CDF Matching Attack [15]	2018	QPR	✗	✗	✗	✓	✗	✗	Uniform*	✓	●
Access Pattern and Search Pattern Attack [16]	2019	FPR	✓	✓	✗	✗	✗	✗	Agnostic	✗	✗
e-Approximate Database Reconstruction Attack [6]	2019	APR	✓	✗	✗	✗	✗	✗	Uniform	✗	✗
e-Approximate Order Reconstruction Attack [6]	2019	AOR	✓	✗	✗	✗	✗	✗	Uniform	✗	✗
Agnostic Reconstruction Attack [17]	2019	FPR	✗	✓	✗	✗	✗	✗	Agnostic	✗	✗