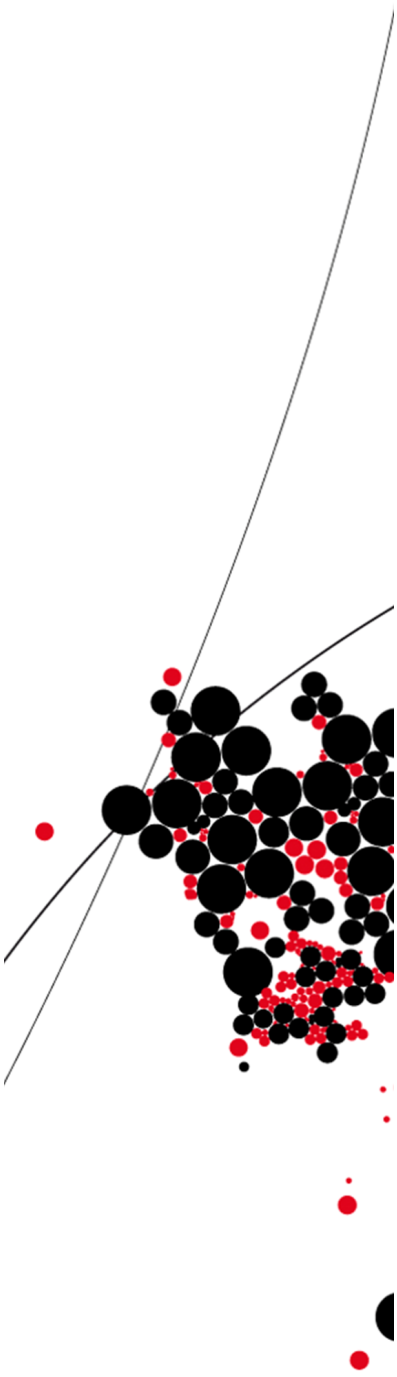
UNIVERSITY OF TWENTE.

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



Active Reconstruction Attacks on 2D Range Databases

Thomas Sierink

MSc Thesis
August 2022

Supervisors:
dr. ing. F. Hahn
dr. A. Sperotto
C. Van den Bogaard, MSc

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

Active Reconstruction Attacks on 2D Range Databases

Thomas Sierink
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
t.g.sierink@student.utwente.nl

***Abstract*—The rise of searchable encryption solutions brings with it a rise of new security needs. In the context of searchable encryption, specifically 2D range query databases, passive reconstruction attacks have been proven to be effective in various situations. This paper set out to discover what extra possibilities an *active* attacker has, and how these might applied to improve chances of fully reconstructing a database. In this study, experiments have been performed on a 30x30 database. These are under the assumption that the scheme has not mitigated leakage and that there is no forward secrecy. Of these methods found, replaying queries and injection show to be capable of query reconstruction. Should the active attacker have access to the encrypted values, there is no additional advantage for them to make use of. Targeted record injection in and close to the databases' corners show an increased chance of full reconstruction, especially after post processing. The average Mean Squared Error per reconstruction is lower for injected databases, improving the expected accuracy should the adversary have to pick a random database from the returned reconstruction set. For a specific injection method, this set is reduced in size by around 10%. Overall, an active attacker has a clear, additional advantage over a passive attacker for all injection methods for a point density of around 50-55 points. This effect is consistent for other square domain sizes besides 30x30, with different relative densities.**

I. Introduction

For the past years, the amount of companies and individuals that make use of cloud storage has been increasing. This growth will continue, as the market is projected to grow 450% from 2022 to 2029 [1]. To ensure security of the data stored, it may be stored encrypted. These remote, encrypted databases offer a great storage capacity, and even provide functionality as some basic operations and queries over the stored data. However, with the need of remote storage increasing, so does the dependence on its security. To aid in secure querying of these databases, searchable symmetric encryption (SSE) [2] classically allows a user to send keyword queries to a server. After this, other schemes emerged supporting multi-keyword queries or range queries over numerical data. Examples of range query schemes are MRQED, a multi-range query scheme, or the works of Kerschbaum et al. and Boneh and Waters [3]–[5]. Several (cloud) security providers such as Ironcore Labs, Crypteron, and Lookout offer applications of searchable encryption [6]–[8].

An important byproduct of searchable encryption is *leakage*. This is the result of the nature of querying databases, where a query is followed by a reply. From both the query and a reply, information can be deduced, therefore referred to as (information) leakage. It may be mitigated, but this is at a cost of efficiency. In the context of encrypted databases there is the minimal assumption that an adversary, e.g. an untrusted server, can indeed observe incoming queries and outgoing responses, all encrypted. Primarily, leakage is either *search pattern* leakage or *access pattern* leakage. An SSE scheme leaks the search pattern if an adversary can identify and thus recognize and differentiate each unique query from another. The access pattern is leaked when an adversary observes a query's respective response, i.e. every data record corresponding to an unknown query.

Using said leakages, SSE schemes can be attacked in several ways. Both databases and queries may be reconstructed [9], [10]. Database reconstruction may be performed over several dimensions i.e. 'columns' of the database. For instance, a 2D database consists of an arbitrary number of records, each holding two values. Reconstructing such a database means to produce databases that share the same leakages as the original database. These reconstructed databases combined form families of equivalent databases, indistinguishable in terms of leakage.

Reconstructions can be either Full Database Reconstruction (FDR) or Approximate Database Reconstruction (ADR). FDR aims to deliver reconstructions, containing each original record. This is the goal, however, not the guaranteed result due to general properties of databases and certain reconstruction methods. ADR does not aim to return a complete copy, but allows an attacker to have a given (un)certainty of the reconstructed databases, and therefore requires less leakage of the original database. Again, such leakage could be the access pattern or search pattern, defined earlier. However, derived information such as the query (frequency) distribution and database size can also aid in particular reconstruction methods [9], [11], [12]. The query distribution helps 'contextualize' the search and access patterns. If the distribution is uniform, for instance, the attacker knows that the access pattern is extracted at a fair distribution. The database size helps in a similar manner: the amount of possible queries depends on the database's domain, for example.

Concerning reconstruction on 2D range query databases, Markatou, Falzon, and Cash have shown methods for FDR (FMC20) [13] and ADR [14]. The leakage required can be a combination of different types and combinations of leakages: (1) the access pattern and known query distribution, (2) the search pattern and known query distribution, or (3) the search pattern and a known database size. These reconstruction methods on 2D range query databases rely on concise calculations and processes, and can be explained geometrically.

This allows it to be visualized clearly too. However, all range query database reconstruction attacks, including the above, assume a *passive* attacker. Therefore, this work studies what implications the extension of the attacker model to an *active* attacker may have. Specifically, this research aims to build upon the FMC20 attack. As such, this paper set out to answer the following question: *To what extent can an active attacker deploy techniques that benefit 2D range database reconstruction?*

By assessing four different active techniques ('Query replay', 'Database manipulation', 'Injection', and 'Fingerprinting'), on randomly generated databases of domain $[30, 30]$ with varying densities, the following insights are achieved.

Insights

- Two targeted injection attacks on databases containing up to 55 records give the adversary an advantage in FDR, reconstructing the database without needing any post-processing. This advantage is higher probability of successful FDR and decreases as the volume of 55 points is approached.
- All targeted injection attacks on databases containing 55 points or less followed by post-processing offer higher probabilities of successful FDR, even compared to post-processing where an existing point in the original database is known.
- Targeted injection attacks on any database will reduce the amount of components to one. This may reduce the size of the set of equivalent databases returned, depending on what points are injected.
- The injection of records not only aids in database reconstruction, but also allows query reconstruction and knowledge to be gained on database alterations.

We first go over related work of attacks and mitigations in the field of database reconstruction attacks in Section II. The preliminaries on 2D range database reconstruction attacks are discussed in Section III, and what possibilities these offer for an active attacker in Section V. Two of these are assessed, of which the results are shown in VI. This research is then discussed in Section VII, and finally the conclusion can be found in Section VIII

II. Related work

Grubbs et al. have developed two 'sacrificial ϵ-ADR' algorithms for range queries [12]. This method only relies on the precision ϵ, not on the amount of records R, making it scale-free. Furthermore, if $\epsilon = \frac{1}{R}$, the algorithm is capable of reconstructing the complete database. This approach extends research by Kellaris et al. [9], who pioneered an FDR in 1D, given access pattern leakage, namely with at least N^4 amount of queries, where N is the 1D domain size. And even better, $N^2 \log N$ for dense databases. It mainly achieves this by ordering the entries. By comparing subsets and supersets, i.e. sets that include or exclude one specific record, that one record is ought to be the closest to a domain endpoint.

Lacharité et al. have improved this further to only needing to observe $N \log N + O(N)$ queries, given that the database is dense [15]. This method sorts the dense database with the use of rank information leakage. The same approach enables an approximate reconstruction attack. Furthermore, Markatou et al. [16] contribute with an FDR attack that makes use of search pattern leakage, independent of query distribution. With the aid of PQ-trees and similar rank information leakage, the order is reconstructed. Then, sets of equations related to the ordering must be solved to fully reconstruct the database with a high probability.

Many other attacks on searchable encryption have been developed, specifically on databases supporting keyword queries. Such attacks can be scale analysis attacks [17], capable of finding the absence or presence a shared keyword among files, keyword guessing attacks [18], capturing valid trapdoors and performs off-line guessing until there is a match. In inference attacks [19] such as IKK attacks [20], the attacker aims to intercept queries and indices, and tries to find a sequence of indices such that these are part of the same query trapdoor. Moreover, chosen document attacks by an active attacker can be applied to recover queries or keywords [21]. Apart from keyword queries, methods for full reconstruction of databases supporting one-dimensional k-Nearest Neighbor queries have been found by Kournapoulos, given that the access pattern contains ordered responses and additional information is available. This approach uses Voronoi diagrams, aiding in the analysis of the database, those supporting k-NN specifically [22]. Later, Kornapoulos presented k-NN reconstruction attacks that are distribution-agnostic and are successful using a scheme's search pattern [23]. It improves an earlier attack with a different Voronoin-based algorithm and employs size estimators and general optimization.

Several methods have been developed to mitigate or reduce various leakages. 'Pancake' for example, applies so-called frequency smoothing to transform access pattern into a uniform, encrypted data storage [24]. Another secondary leakage based on the access pattern is the number of responses given a query: volume. Patel et al. define definition of volume-hiding leakage functions [25]. Furthermore, SEAL is a family of devised SSE schemes with adjustable leakage, to be defined at setup [26]. Besides mitigating leakage, forward secrecy is a mitigation that prevents file-injection attacks, and can help against the replaying of queries [27], [28]. After an alteration of the database, queries created beforehand are no longer compatible with the database.

III. Database Reconstruction

In this section, we first discuss the underlying theories and concepts that aid the FMC20 attack, before diving into the active attacks that could aid this database reconstruction.

2D databases can be easily visualized as a grid, where each integer point on the grid represents a valid possible entry in the database. As such, a database `DB` can be described as a 2-dimensional domain D with

$$D = [N_0] \times [N_1], \qquad \text{where } [N_i] = \{1, 2 \ ... \ N_i\}$$

If the domain sizes of D are equal, $N_0 = N_1$, we say that D is *square*. Furthermore, we define a point on the grid $w =$

(w_0, w_1). The main diagonal of any grid is the line from $(0, 0)$ to $(N_0{+}1, N_1{+}1)$. Mind that this means that the database does not span the entire grid: the grid has domain $[0, N_i{+}1]$ in the i-th dimension, whereas the database has domain $[1, N_i]$. This is visualized in Figure 1, where the database domain is confined by bold grid lines. We define the database DB over domain D as a R-tuple of integer points lying on the grid points of D. These 2D integer points are referred to as records. Each record has a matching ID $j \in [R]$. Therefore, the record's values are $\texttt{DB}[j]$. Figure 2 shows an example of a 10-by-10 database visualized as such a grid.

Database density, or how 'filled' a database is, is a concept that is hard to grasp in qualitative ways. Generally, literature speaks of 'dens(er)' or 'spars(er)' without speaking of exact density. A general notion of when a database is purely 'dense' does exist. A database is dense when every domain value of each dimension is found in at least one record. For matrices in mathematics, density equals the ratio between the number of records in the database divided by total amount of unique possible points within the database.

An important concept that is part of the FMC20 attack is *dominance* and *anti-dominance*. It shows the positional relationship between points. A point $w \in D$ dominates $x \in D$ if w is to the top right of x. We say $w \succeq x$. Anti-dominance does *not* mean the exact opposite. Rather, a point $w \in D$ anti-dominates $x \in D$ if w is to the top left of x. We say $w \succeq_\alpha x$, of which an example is shown in Figure 1.

Definition III.1 (Dominance and anti-dominance)**.**

$$w \succeq x \ : \ w_0 \geq x_0 \wedge w_1 \geq x_1$$

$$w \succeq_\alpha x \ : \ w_0 \leq x_0 \wedge w_1 \geq x_1$$

As previously mentioned, database DB supports range queries. Examples of queries can be found in Figure 2, returning all points matching the queries' domains [2-4]x[3-8] and [5-10]x[8-10].

Definition III.2 (Range queries)**.**

$$q = (c, d) \in D^2 \ : \ d \succeq c$$

Subsequently, the database responses with all entries matching the query.

Definition III.3 (Response)**.**

$$\text{Resp}(\texttt{DB}, q) = \{j \in [R] \ : \ d \succeq \texttt{DB}[j] \succeq c\}$$

The response multiset of DB, RM(DB), is the multiset of responses to all possible queries q_i, meaning the multiset of the entire access pattern. Since this multiset may contain queries returning the same records, we also define the response set RS.

Definition III.4 (Response multiset and response set)**.**

$$\text{RM}(\texttt{DB}) = \{ \ \{\text{Resp}(\texttt{DB}, q_i)\} \ \}$$

$$\text{RS}(\texttt{DB}) = \text{set}(\text{RM}(\texttt{DB}))$$

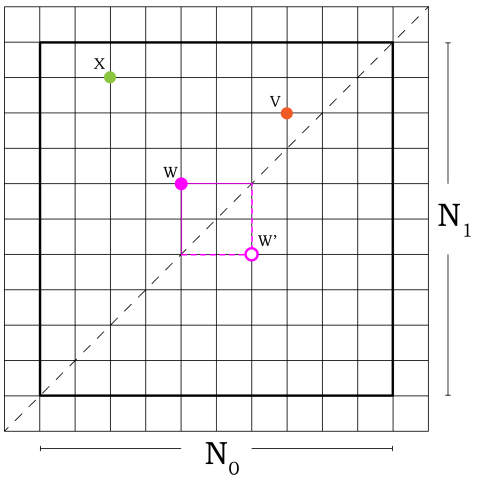


Fig. 1. Example of a grid visualization of a database with sizes N_0 and N_1. w' is the reflection of w. v dominates w, x anti-dominates w and v.

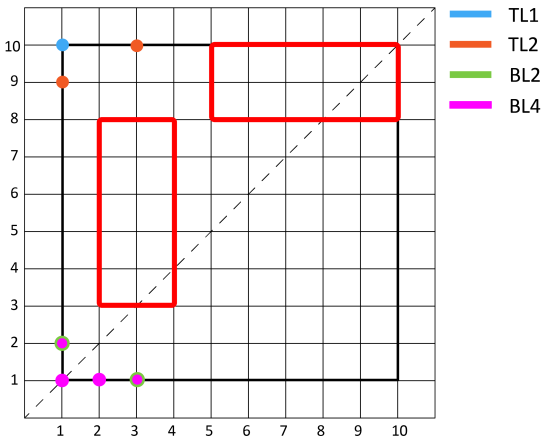


Fig. 2. Example of a 10-by-10 database and queries.

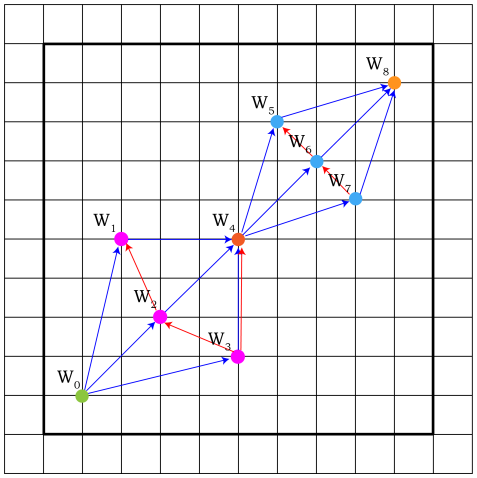


Fig. 3. Example of a dominance graph (blue arrows) and an anti-dominance graph (red arrows). The canonical antichain partition consists of antichains $\{\{w_0\}, \{w_1, w_2, w_3\}, \{w_4\}, \{w_5, w_6, w_7\}, \{w_8\}\}$

Related to dominance are *chains* and *antichains*.

Definition III.5 (Chains and graphs)**.** A subset of points S from grid D form a chain if any point is dominant with respect to another point. An antichain is a collection of points where any point is anti-dominant to any other point. A point's *height* is the length of the longest chain where it is 'most (anti-)dominant'. The collection of points of the same height make up a partition in the form of an anti-chains: the canonical antichain partition. An example of this can be found below in Figure 3.

Another metric used for database analysis is the *query density* of a point x in D. This number represents in how many unique queries this point is included in the response, i.e. lies within the query domain.

Definition III.6 (Query density)**.**

$$\rho_x = |\{(c,d) \in D^2 \ : \ d \succeq x \succeq c\}|, \ \ x \in D$$

This can be geometrically determined by:

$$\rho_x = x_0 x_1 (N_0 + 1 - x_0)(N_1 + 1 - x_1)$$

For a pair of points (x, y) where $y \succeq x$:

$$\rho_{x,y} = |\{(c,d) \in D^2 \ : \ d \succeq x, y \succeq c\}|, \ \ x, y \in D$$

$$\rho_{x,y} = x_0 x_1 (N_0 + 1 - y_0)(N_1 + 1 - y_1)$$

Definition III.7 (Reflection)**.** The reflection (along the main diagonal) of w, w' or $\sigma(w)$ expressing reflection as a function, is calculated as

$$\text{reflected point } w' = (w'_0, w'_1)$$

$$w'_0 = w_1 \cdot \frac{N_0 + 1}{N_1 + 1}, \ \ w'_1 = w_0 \cdot \frac{N_1 + 1}{N_0 + 1}$$

A database can be divided into components, of which an example can be seen in Figure 4. Such a component contains all points in the database that themselves or their reflections both dominate or are dominated by all other points of the database that are not in the same component. Any database can uniquely be divided into components. This does not mean that there is a one-to-one correspondence, however. Furthermore, reflecting any number or combination of components will give an equivalent database.

Definition III.8 (Components)**.**

$$q \succeq p, \sigma(p) \vee p, \sigma(p) \succeq q, \ \ \forall p \in C, \ \ \forall q \notin C, \ \ p, q \in \texttt{DB}$$

With the knowledge gained by the above tools, concepts and calculations, the result will not be one single reconstructed database. As will now be explained, after reconstruction, there exist families of equivalent databases with the same properties.

A reconstructed database is not necessarily an exact copy of the original database. Several databases can share the same leakage and are in that regard indistinguishable. These databases are referred to as *equivalent databases*, and make up the set $E(\texttt{DB})$. Such a set is also returned by a reconstruction attack.

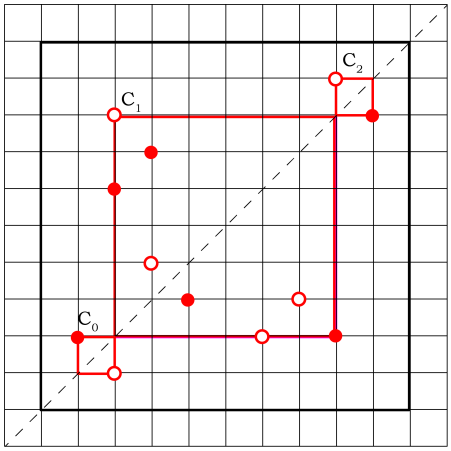


Fig. 4. Example of components in a database. Each component C_i may be reflected, resulting in an equivalent database.

Definition III.9 (Equivalence with response to the response multiset)**.** Two databases DB and DB′ are equivalent if and only if RM(DB) = RM(DB′). In other words, DB and DB′ are both within $E(\texttt{DB})$.

A database rotated or flipped, following the eight rigid motions of a square remains equivalent to the original database with response to the multiset. One can imagine that manipulating the grid in this way keeps all relations between the records intact. Alternatively, one could also imagine all queries to be flipped. This would then still result in the same leakage pattern, since all possible queries are still queried, purely by geometry: every possible rectangle, line and point on the grid is a valid query.

Another way to create equivalent databases with respect to the response multiset is the reflection of points and components. It is important to remember that FDR makes use of this multiset rather than the set. Only equivalencies with respect to the multiset are indistinguishable by the attack.

IV. Attacker Model

In cyber security research, it is key to have a clear, concise attacker model for which all findings are based on. A scheme or technique is only defined to be secure with respect to the context it is designed in. On a high, abstract level, an attacker is either active or passive.

A passive attacker, be it an observer of all traffic or a malicious server, merely observes, and does not modify, replay, or send. The opposite is true for an active attacker. An active attacker may perform several actions, such as spoofing, replay attacks, or Denial-of-Service attacks. This scenario is closely related to that of Multi-Party Computation, where users together interact with a server to perform operations rather than queries on encrypted data, as described by Oded Goldreich [29].

Let us now redefine the notions of passive and active in the context of database attacks. The definition for a passive attacker can be left untouched: a passive attacker observes queries by users, replies, and local encrypted values, can store this data and perform operations on these, such as computing a

response multiset. An active attacker can do more. Suppose the active attacker has direct access to the encrypted database. The injection of records in the database would fit the description, just like the chosen document attack by Cash et al. [21], and so is the act of replaying queries to the database, assuming the lack of forward secrecy. Forward secrecy would prevent this, since newly added records are not valid with previously created queries. Locally, however, nothing would prevent this adversary to manipulate the database, like deleting, switching or copying values, or transforming the data in any other way.

To put this attack in context, a common example used in attacks on 2D range databases is geo-location, where the records hold longitude and latitude. However, any two integer properties are valid. Non-integer values can be multiplied by a factor of 10^x, where a higher integer x will offer a higher precision, since all decimals are removed.

In the next chapter, we will go over how the model and the shift from passive to active might allow new or improved techniques aiding FMC20.

V. Attacks

A. Query Replay

Given that there is no forward secrecy implemented in the database, an adversary would be able to record and save observed query tokens and re-query the (altered) database with these. In the case of absence, the adversary can gain information in various ways. First, imagine that the database is an offline copy: the adversary knows that there will be no modifications to the database. Injecting a point (a,b) in the database, followed by a query using a saved token, allows the adversary to find information on the query's search range. The process would be as follows.

1) Query the database using a query as described in Definition III.2, with query bounds $c_{lower}, c_{upper}, d_{lower},$ and d_{upper}
2) Record all returned IDs.
3) Inject record (p, q).
4) Query the database with the same token again.
5) Compare the set or returned IDs with the previous set.
   a) Are the sets the equal?
      $p \leq c_{lower} \vee c_{upper} \leq p \vee d_{lower} \leq c \vee d \leq d_{upper}$
   b) Are the sets unequal i.e. the injected point is included, then $c_{lower} \leq p \leq c_{upper} \wedge d_{lower} \leq q \leq d_{upper}$.

Using this, the attacker can iteratively gain more and more knowledge on the limits of the query. The determining of a query's range is out of this paper's scope, from a first view. However, after gaining knowledge on query ranges, there are implications for database records as well, besides information about records that lie within the now known range. For example, should the database be updated with new data, then recognizing the addition of a new point and its approximate coordinates is very powerful. For instance, when keeping up with real-time location data.

The replaying of queries is of use in the injection attacks. The queries can be recorded before implementing the injection attack, meaning that the adversary does not need to wait for the entire new injected access pattern to be observed again. They could just re-query the entire database themselves.

This query reconstruction attack is different compared to earlier file-injection attacks, where a file containing keywords of interest is injected in an encrypted database [30]. This 2D range query method means that injection can be more approximate, since the injected point can be anywhere within the query range, opposed to being an exact keyword match. Moreover, this approach is based on recursively narrowing down on the query's exact range, with each injection gaining information on the query's range.

B. Database manipulation

Following the goal of easing reconstruction, or to guarantee correctness of reconstruction, the attacker can pre-process the database, by manipulating existing records. Essential about database manipulation is that there is no new information added. These manipulations are under the assumption that the adversary has access to the local encrypted values in storage, for example. From here, there are several ways the adversary might try to pre-process the data.

Firstly, the x- and y-values can be swapped in this situation. This will lead to one of the following two cases. If the database is square, then this will have no practical effect. As stated in and after definition III.9, all rigid motions of the square are equivalent in terms of leakage, and flipping a square diagonally is one of these motions. If the database is not square, then the symmetry axis is not along the line $x = y$, but rather the line from (1,1) to (x_{max}, y_{max}). Flipping values of such a non-square database will then not be an equivalent database anymore. All reconstructions based on this database can subsequently no longer be assumed to also be reconstructions of the original.

Secondly, an adversary could copy one value to the other column. This would project a point to the line x=y vertically or horizontally, depending on whether the x-value or y-value has been copied, respectively. This method will not aid in reconstruction as this simply removes a dimension, and still the database might not be correctly reconstructed. The relationships between all the copied points will have been apparent in the response multiset in the first place. Note that copying values to other rows deletes information as well, creating a whole new point. The original value that has been replaced cannot be recovered again. A solution to this would be to create two copies of the database, one containing the x-values, and the other y-values. However, still no advantage is gained.

C. Injection

As mentioned in Section III, an attacker attempting to reconstruct the database will retrieve a family of equivalent databases, $E(\mathtt{DB})$, containing databases that share the same access pattern as the original database. The attacker will have no knowledge whether a database amongst these is correct, and if so, which database this is. The idea of record injection is to merely increase the attacker's chances of indeed reconstructing a database correctly, i.e. the fully reconstructed database is one of the reconstructions returned. For now we only look

at the injection of points and the resulting output of the reconstruction process. In the following subsection, we look at increasing chances with an extended injection attack referred to as 'fingerprinting'.

As mentioned in the preliminaries, a database consists of one or more components. These components are parts of the database that can be individually reflected in any configuration. These reflections do not alter the response multiset, keeping the reconstruction valid relative to the original database. Since the attacker still needs to check every returned database for correctness, it is wise to reduce the size of $E(\texttt{DB})$, for instance by reducing the amount of reflectable components. To eliminate components, the attacker can attempt alter the database such that there is only one component left. This is done by merging all components into one single component, fixing the amount of possible databases ($E(\texttt{DB})$) returned based on component reflections from 2^n to 2, where n is the amount of components. As can be seen in Figure 5, based on the original database of Figure 4, instead of 8 (2^3) reflection possibilities, all components are merged into one single reflectable component C'$_0$ by injecting a point in the top left: $[1, y_{max}]$, later referred to as TL1. Alternatively, a combination of one or multiple points of the following properties will merge all components. To achieve a single component by injection, either one point must lie at $[1, y_{max}]$, or at least one point must lie on the line from $[1, y_{max} - 1]$ to $[1, \lceil y_{max}/2 \rceil]$, **and** one on the line from $[2, y_{max}]$ to $[\lceil x_{max}/2 \rceil, y_{max}]$.

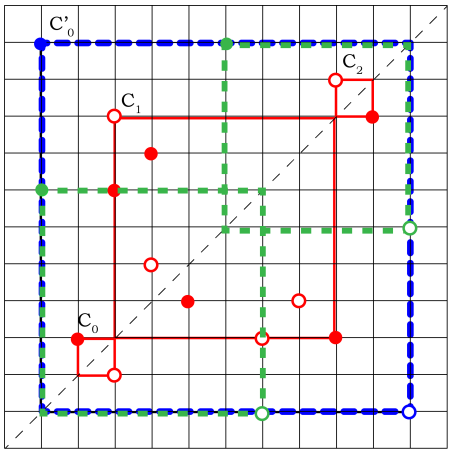


Fig. 5. Injected database with a point in the top left, merging the three original components indicated by the red, straight squares, into one single component, the blue dashed square. Alternatively, a set of overlapping components

Furthermore, the injection of random points to the database could be of help to introduce more information, contributing to more information between points to be found in the access pattern. It improves the density of the database, theoretically improving the accuracy of reconstruction as shown by Markatou et al. [13].

D. Fingerprinting

The reconstruction attack returns various database reconstructions, and at face value, the attacker has no way to tell which one is correct. With knowledge of record values, the attacker can check the reconstructions for the appearance of such a point. However, this does not give guarantees of the database being an exact match. Recalling the fact that a database is equivalent through every rigid motion of the square, a different record might land on the exact point, giving the attacker a false positive. Realizing that there is no guarantee of knowing a point already existing within the database, injecting known points offers a solution here as well. After the injection of these points, the attacker can then rotate or transform the reconstruction until all known points that are injected are matched. Moreover, injecting points in an asymmetric pattern allows the attacker to eliminate incorrect symmetries of the database. This method will be tested using the L shaped set of points $[(1,1),(1,2),(2,1),(3,1)]$ (BL4) and a reduced set $[(2,1),(3,1)]$ (BL2) and $[(1, y_{max}-1),(3, y_{max})]$ (TL2). A set such as $[(1,1),(1,2),(2,1)]$ will be less ideal, since this triangle shape is symmetric across the diagonal of a square database and will therefore not be used. In Figure 2, these injection 'shapes' can be found.

VI. Results

In this section, we will look at the experimental results found for attacks V-C and V-D. I would like to acknowledge the efforts of Dominique Dittert and Janes Rausch from TU Darmstad, who wrote an *FMC20* extension to the open-source framework '*LEAKER*' [31], enabling this research. It assumes the entire access pattern is known. The code used to obtain these results is not the same code as performed by Markatou et al. The LEAKER framework excluding the FMC20 attack can be found on Github, as well as the code used to obtain the results. [1] [2] All methods have been tested on 100 randomly generated databases with a [30,30] domain of increasing densities, 5 through 100 points. The domain size was chosen for practical reasons, as required memory storage increases rapidly with the domain. Figure 6 shows the memory usage and runtime for a single database reconstruction attack with minimal and maximal density, 5 and 100 records respectively. These results are consistent with other 2D range query reconstruction testing [32]. Furthermore, the injection methods are compared to random control methods: the injection of 1 through 4 random points (RI1, RI2, RI3, RI4), and the knowledge of 1 through 4 randomly known existing points (K1, K2, K3, K4).

Of these methods, several metrics will be presented in this section. Interesting to look at first is the amount of successfully reconstructed databases, the reconstruction rate. It shows whether a set of reconstructions is returned of which at least one database is a fully correct reconstruction. This metric divided by 100% gives us a secondary metric of the success rate.

When we look at the amount of successfully reconstructed databases in Figure 7, meaning excluding post-processing, we observe a clear trend. K1-4 and the 'original' scores fall perfectly in line, since these are the exact same databases and there is no post-processing step making use of the extra information. Apart from that, we observe that two injection methods, BL2 and BL4, improve the chances of reconstructing

[1] https://github.com/encryptogroup/LEAKER
[2] https://github.com/TSierink/ActiveFMC20/

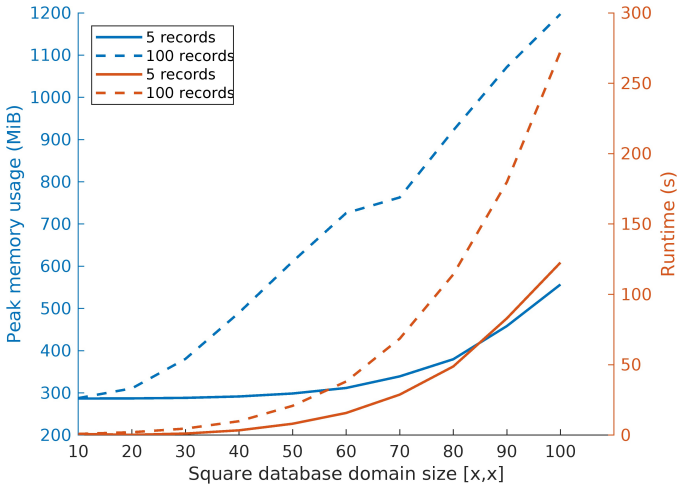


Fig. 6. Runtime and peak memory usage for varying domain sizes and record numbers for a single reconstruction attack using LEAKER's FMC20 implementation.

the database given that it is sparser than about 50-55 records. TL1 and TL2 do not show this behaviour. These perform consistently worse, to later converge with the general trend as the database grows denser. The true additional advantage of injection versus a known point lies within the area between the injection methods and K1-4. This means that up to 50-55 points in this specific database domain, injection of a record is more effective than knowledge on an existing record, and the 'original' passive attack.

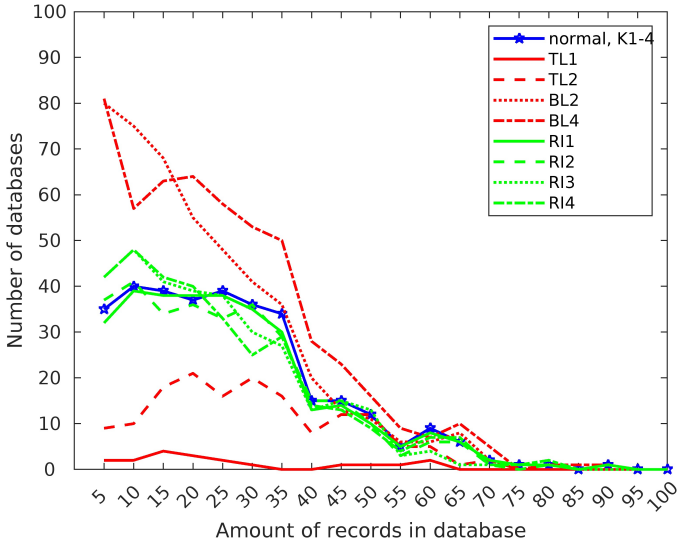


Fig. 7. Correctly reconstructed databases for each method, for increasing database density *without* post-processing.

Should the adversary not be able to apply post-processing, they would need to handle a brute force approach of picking a random reconstruction. To assess the expected error of this randomly chosen database, Figures 8 and 9 show the average Mean Squared Error (MSE) per reconstruction per database. These results show that both the location of injection and the amount of injected points contribute to a reconstruction's MSE. The injection methods all show large values for sparser databases. This was to be expected, as the injection of points in a sparse database has more impact. Namely, to be inaccurate on one in five points is more impactful than one in a hundred. Furthermore, recall that reconstructions could be any rigid motion of the square. Since the points are injected in a corner, they are have the greatest distance from the midpoint. Thus, after a rotation or reflection, they find themselves further from their original location, explaining BL4's inaccuracy. More concretely, for sparsely densed databases with up to 40 records, we see that K1-4 have the lowest average MSE per database in the returned $E(\mathtt{DB})$. Generally for denser databases, we find that the average is slightly lower than the regular reconstructions, with TL1 performing best, and BL2 worst for dense databases.

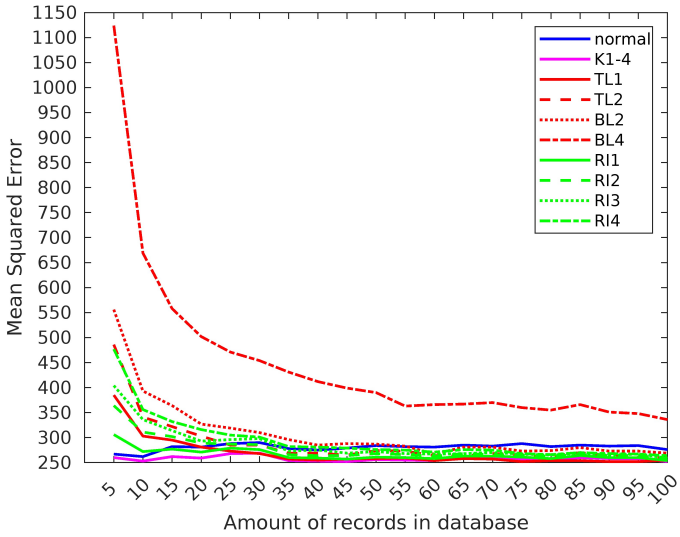


Fig. 8. Zoomed-out view of the MSE for each method, for increasing database density.

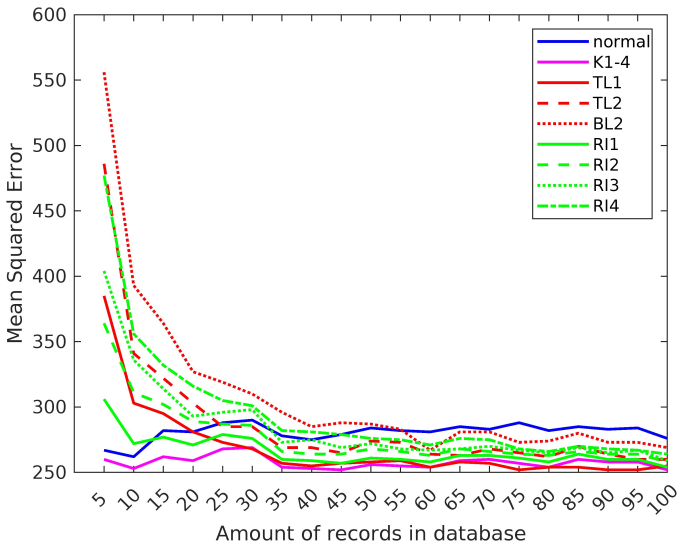


Fig. 9. Zoomed-in view of the MSE for each method (excluding BL4), for increasing database density.

Another factor contributing to the 'brute-forceability' and ease of attack is the amount of equivalent reconstructions being

returned. As theorized in Section V-C, injecting points in the top left will reduce the amount of components i.e. the size of $E(\texttt{DB})$, from its maximum of 2^n, where n denotes the number of components, to just 2. Take not that there is still an additional factor of 8 for each rigid motion. The changes in $|E(\texttt{DB})|$ can be observed in Figure 10 where on average, the attack returns around 5 reconstructions after TL2 injection, consistently lower than the original database's reconstruction. This shows behaviour as theorized in attack V-C. It must be noted, again, that the returned set is not guaranteed to hold a correct reconstruction. The difference in the results of TL2 and TL1 lies in the fact that a corner point such as TL1 can be rotated and possibly still suit the access pattern. For example, imagine a database where each corner contains a point. A similar database with asymmetric 'figures' in each corner will not have the same access pattern when rotated. Moreover, small scale experiments show that for any pair of points merging the components, the size of the returned set is lower than the uninjected database.

BL4 adds reconstructions compared to no injection. For BL4, this difference of 1 can be traced to the fact that the figure in itself can form one single component, given there is no point lying above its entirety.

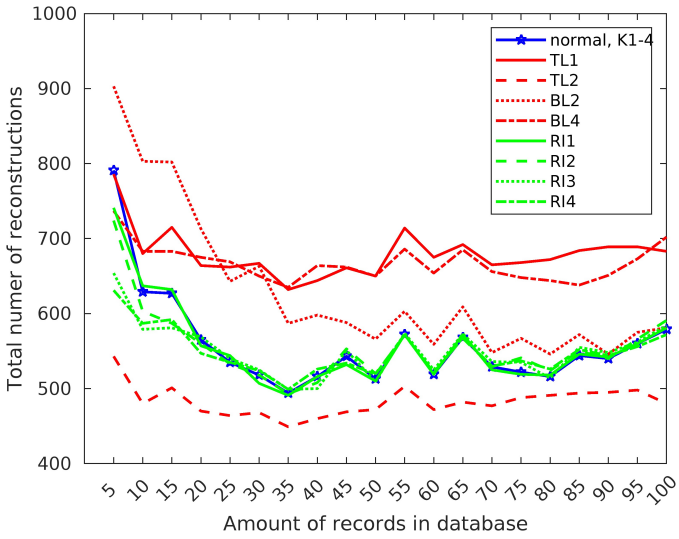


Fig. 10. Number of total reconstructions per 100 iterations of a random database for each method, for increasing database density.

As shown in the final results in Figure 11, post-processing has a clear positive effect on database reconstruction. It can be seen that there is a distinction between three groups. Firstly, regular passive reconstruction, which is the same as the curve in Figure 7, as this control case does not have any known or injected points post-processing can be applied to. The middle group contains all the reconstruction rates of the methods that either have knowledge on 1-4 randomly chosen points, or that have injected 1-4 random points. The scores of K1, K2, K3 and K4 coincide. The third group is made up from the custom methods: TL1, TL2, BL2 and BL4. All these methods converge at a density of 25-30 points. Gradually, the 'regular' control method drops to a reconstruction rate of 0%, whereas the others approach or equal 100%. The targeted injected methods get to this 100% fastest. The true additional advantage of injection versus a known point lies within the area between the injection methods and K1-4. This means that up to 50-55 points in this specific database domain, injection is more effective than knowledge on a record. This limit of advantage between known points and injected points for various domain sizes is shown in Figure 12. Here, the square root of the amount of points, or density is plotted against domain sizes. This straight line shows the square relationship between domain size and density, meaning that the limit of said advantage is linear with the database's density.

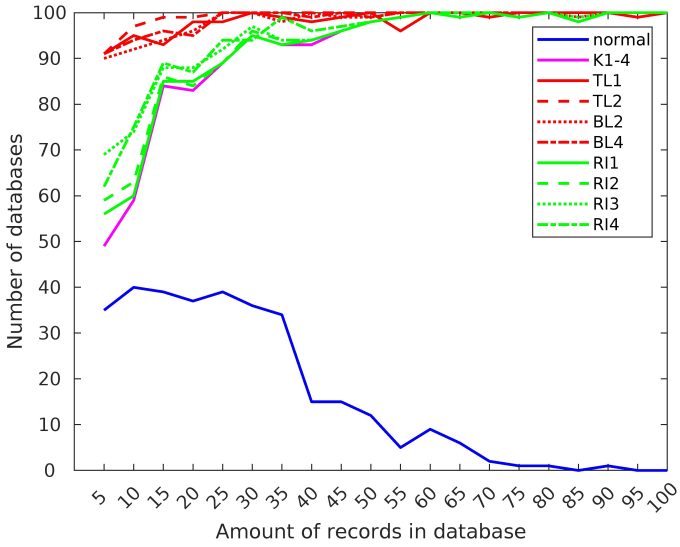


Fig. 11. Correctly reconstructed databases for each method, for increasing database density *with* post-processing.

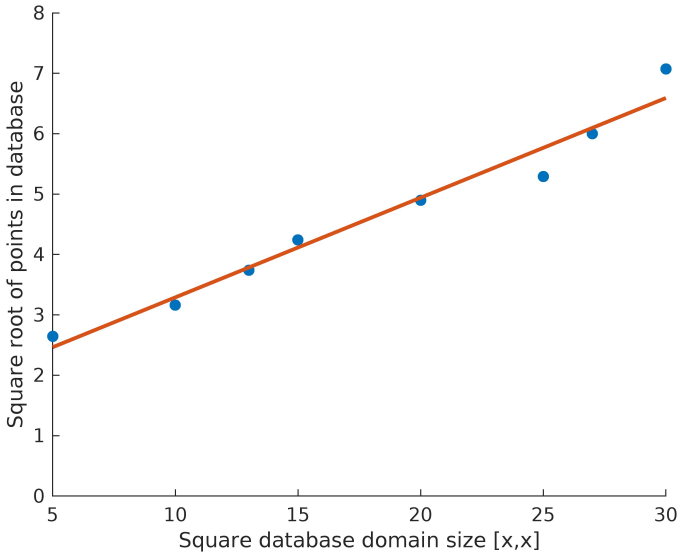


Fig. 12. Linear relationship between the square roots of highest number of points where injection is more effective than knowledge on points for different square domain size.

VII. Discussion

On several points, remarks are necessary to either nuance the findings, or sketch a clear context wherein the results were found. Firstly, the 'LEAKER' code used to reconstruct databases used assumes that every response for every query is known. This means that the access pattern must be complete, including the set of empty responses. The latter could be regarded as unconventional, since the access pattern generally only contains the responses, not lack thereof. Should there be mitigations in place that prevent the leakage of the access pattern, then FMC20 is feasable. Moreover, the active attacks depend on the absence of forward secrecy. Furthermore, this library is not an exact copy of what FMC performed in their own paper, it is an implementation. Runtime and memory usage could differ, and exact results could differ. However, both relative effects still hold. Thirdly, targeted injection attacks in the top left, top right, or bottom right, require some knowledge of the database's domain, making this an attack not solely based on leakage. This could be roughly approximated by either the size of the search pattern or the size of the access pattern. Lastly, the variability between several test runs of 100 iterations still show a 5-10% variability in reconstruction rate. These hiccups can be seen in the results, but would decrease by volume. Most of all, this proves how much the specific layout of points on the grid determines the ease of reconstruction, or for example the amount of components.

VIII. Conclusion

An active attacker can deploy several techniques based on the knowledge and access they have. Firstly, the recording and repeating of queries discussed in Section V-A could be very interesting and powerful in both learning about queries and the addition of points in the database.

As an active attack, database manipulation found in Section V-B does not provide an advantage to the active attacker. Either the database is altered in such a way that it is no longer equivalent, or information is lost and is not retrievable after reconstruction.

'Fingerprinting' as discussed in Section V-D shows to be an effective strategy to more reliably reconstruct sparsely densed databases, especially when points can be injected in the database's corners. This requires additional information on the database's domain, apart when injecting the bottom left corner. Without the post processing of Section V-C, only points injected in the bottom left and bottom right corner are more effective. These points are dominated by, or dominate all existing points, respectively. This lengthens dominance chains, and furthermore introduces dominance relations for every existing points. Future work must show whether these database concepts do indeed directly improve database reconstruction. Random point injection has a similar effect to an attacker's chances of reconstructing as knowing an existing point within the database, post and pre processing. The difference is that it introduces more information on point's location, meaning that incorrect reconstructions are more incorrect too, as there are more points to be mistaken.

In terms of number of reconstructions, the size of set $E(\texttt{DB})$ the attacker needs to check for possibly correct reconstructions, only TL2 and other multi-point injections as described in subsection V-C reduce the amount of reconstructions returned.

For future work, different domain sizes could be tested. This allows for experimenting on larger empty spaces within a database, where an injection might be even more powerful. Moreover, optimizing would help researching these larger domains. The memory usage and run-time are all related to the number of unique queries exist for a database. However, with knowledge on the presence and absence of points, or perhaps neglecting points, not all regions would need to be queried. Furthermore, future work could test more chosen injection configurations, or develop completely different strategies. Ideas include a contextual approach, using probabilities connected to possible point locations combined with approximators as seen in ADR, but with known points as input. Another idea is to inject or use known non-corner points as new corners of smaller 'sub-databases', which result from splitting the grid up based on these points. Lastly, interesting future work lies in determining feasibility of these actives approaches in multidimensional reconstruction attacks, because the current generic multi-dimensional attack does not rely on the access pattern.

References

[1] Fortune Business Insights, "Cloud storage market size, share & covid-19 impact analysis," 2022. [Online]. Available: https://www.fortunebusinessinsights.com/cloud-storage-market-102773

[2] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, 2000, pp. 44–55.

[3] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007, pp. 350–364.

[4] F. Kerschbaum and A. Tueno, "An efficiently searchable encrypted data structure for range queries," *CoRR*, vol. abs/1709.09314, 2017.

[5] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of the 4th Conference on Theory of Cryptography*, ser. TCC'07. Berlin, Heidelberg: Springer-Verlag, 2007, p. 535–554.

[6] IronCore Labs, "Encrypted search - ironcore labs," https://ironcorelabs.com/docs/data-control-platform/concepts/encrypted-search/, 2021, (Accessed on 02/28/2022).

[7] Crypteron, "Crypteron introduces secure, searchable encryption : Crypteron," https://www.crypteron.com/blog/practical-searchable-encryption-and-security/, 2020, (Accessed on 02/28/2022).

[8] Lookout, "lookout-casb-platform-overview-wp-us.pdf," https://www.lookout.com/documents/whitepapers/us/lookout-casb-platform-overview-wp-us.pdf, 2021, (Accessed on 02/28/2022).

[9] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1329–1340. [Online]. Available: https://doi.org/10.1145/2976749.2978386

[10] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Inference attack against encrypted range queries on outsourced databases," New York, NY, USA, p. 235–246, 2014. [Online]. Available: https://doi-org.ezproxy2.utwente.nl/10.1145/2557547.2557561

[11] P. Grubbs, M.-S. Lacharite, B. Minaud, and K. G. Paterson, "Pump up the volume: Practical database reconstruction from volume leakage on range queries," New York, NY, USA, p. 315–331, 2018. [Online]. Available: https://doi.org/10.1145/3243734.3243864

[12] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson, "Learning to reconstruct: Statistical learning theory and encrypted database attacks," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1067–1083.

[13] F. Falzon, E. A. Markatou, Akshima, D. Cash, A. Rivkin, J. Stern, and R. Tamassia, *Full Database R+ econstruction in Two Dimensions*. New York, NY, USA: Association for Computing Machinery, 2020, p. 443–460. [Online]. Available: https://doi.org/10.1145/3372297.3417275

[14] E. A. Markatou, F. Falzon, R. Tamassia, and W. Schor, "Reconstructing with less: Leakage abuse attacks in two dimensions," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2243–2261. [Online]. Available: https://doi.org/10.1145/3460120.3484552

[15] M.-S. Lacharité, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 297–314.

[16] E. A. Markatou and R. Tamassia, "Full database reconstruction with access and search pattern leakage," in *Information Security - 22nd International Conference, ISC 2019, New York City, NY, USA, September 16-18, 2019, Proceedings*, ser. Lecture Notes in Computer Science, Z. Lin, C. Papamanthou, and M. Polychronakis, Eds., vol. 11723. Springer, 2019, pp. 25–43. [Online]. Available: https://doi.org/10.1007/978-3-030-30215-3_2

[17] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 222–233, 2014.

[18] J. W. Byun, H. S. Rhee, H.-A. Park, and D. H. Lee, "Off-line keyword guessing attacks on recent keyword search schemes over encrypted data," in *Proceedings of the Third VLDB International Conference on Secure Data Management*, ser. SDM'06. Berlin, Heidelberg: Springer-Verlag, 2006, p. 75–83. [Online]. Available: https://doi.org/10.1007/11844662_6

[19] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 644–655. [Online]. Available: https://doi.org/10.1145/2810103.2813651

[20] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *NDSS*, 2012.

[21] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, Oct. 2015. [Online]. Available: https://doi.org/10.1145/2810103.2813700

[22] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "Data recovery on encrypted databases with k-nearest neighbor query leakage," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1033–1050.

[23] ——, "The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1223–1240.

[24] P. Grubbs, A. Khandelwal, M.-S. Lacharité, L. Brown, L. Li, R. Agarwal, and T. Ristenpart, "Pancake: Frequency smoothing for encrypted data stores," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2451–2468. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/grubbs

[25] S. Patel, G. Persiano, K. Yeo, and M. Yung, "Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 79–93. [Online]. Available: https://doi.org/10.1145/3319535.3354213

[26] I. Demertzis, D. Papadopoulos, C. Papamanthou, and S. Shintre, "SEAL: Attack mitigation for encrypted databases via adjustable leakage," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2433–2450. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/demertzis

[27] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, J. Pieprzyk, and L. Xu, "Forward and backward private dsse for range queries," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 328–338, 2022.

[28] J. Wang and S. S. M. Chow, "Forward and backward-secure range-searchable symmetric encryption," in *POPETS*, 2022, p. 28–48.

[29] O. Goldreich, *Foundations of cryptography: Basic applications volume 2*. Cambridge University Press, May 2004.

[30] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of File-Injection attacks on searchable encryption," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 707–720. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/zhang

[31] S. Kamara, A. Kati, T. Moataz, T. Schneider, A. Treiber, and M. Yonli, "Sok: Cryptanalysis of encrypted search with leaker &x2013; a framework for leakage attack evaluation on real-world data," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, 2022, pp. 90–108.

[32] F. Falzon, E. A. Markatou, Z. Espiritu, and R. Tamassia, "Attacks on encrypted range search schemes in multiple dimensions," Cryptology ePrint Archive, Paper 2022/090, 2022, https://eprint.iacr.org/2022/090. [Online]. Available: https://eprint.iacr.org/2022/090