SSE Is Not As Secure As It Looks

New Attacks On Range Queries Using PQ-Trees And Auxiliary Information

J.C.H. Thomas

Delft University of Technology

TU Delft

SSE Is Not As Secure As It Looks

New Attacks On Range Queries Using PQ-Trees And Auxiliary Information

by

J.C.H. Thomas

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday December 14, 2022 at 14:15.

Student number:	5397863	
Project duration:	January 17, 2022 – December 14, 2022	
Thesis committee:	Prof. G. Smaragdakis,	TU Delft, thesis advisor
	Dr. K. Liang,	TU Delft, supervisor
	Dr. S. Roos,	TU Delft
Daily Co-Supervisor :	H. Chen,	TU Delft
External supervisor :	Dr. Ing. F. Hahn,	TU Twente, supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/

TU Delft

Preface

Dear reader, you are about to on the thesis project I have been working on for the past 11 months. It ends my career as an academic student at TU Delft. I enjoyed the experiences I have had during my time as a student and have grown a lot.

I am proud to present the thesis project *New Attacks On Range Queries Using PQ-Trees and Extensions Using Auxiliary Information*. I would like to thank Dr. Kaitai Liang , Dr. Ing. Florian Hahn and PhD. Student Huanhuan Chen for the conversations and feedback which have pushed this research to get the results it has. Furthermore, I would like to thank Prof. Georgios Smaragdakis for being my Thesis Advisor to guide it all in the right processes. Lastly, I would like to thank Dr. Stefanie Roos for being part of my thesis committee and putting in the effort to review it.

J.C.H. Thomas
Delft, December 2022

Abstract

In a world where more data gets uploaded to the cloud, it is essential that the data gets stored securely. For users to keep search functionality, searchable symmetric encryption has been developed. SSE works by a user sending a token representing a keyword (or a range), after which the server returns the documents that match the keyword (or values in the queried range). These observed documents are called the access pattern. A number of attacks that work on range SSE schemes have been developed. Yet, most of these attacks work on density or query assumptions that hinder their performance if these assumptions are false. Furthermore, a number of these attacks use auxiliary information. Yet none of them uses known search tokens.

We, thus, propose a novel approach that uses a PQ-Tree to get the order of the observed documents. It uses the known pairs to assign possibilities for these documents and returns a list of options for each observed token and estimated values for each document. The basic attack guarantees that the correct query is always available by assigning document values based on the known tokens. A refined attack was made that assigns more known tokens by adding the best matches based on confidence score. This refinement allows for more exact token and query matches. Both of these attacks were extended using partial or similar documents from which document volume or rank information could be gathered to further increase the attack performance in cases where such data is available. We evaluate our attacks against current state-of-the-art [28, 27, 37] and outperform those regarding document and query recovery on all tested datasets. Lastly, we use two countermeasures [42] and see that all attacks are impacted but that our attack still outperforms most.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
SSE	Searchable Symmetric Encryption
CSP	Cloud Service Provider
PII	Personally Identifiable Information

Symbols

In our setup we have a maximum of m documents and maximum of n queries.

Symbol	Definition
Q	Query Set, $Q = q_1, q_2, ..., q_{n-1}, q_n$
T	Token Set, $T = t_1, t_2, ..., t_{n-1}, t_n$
R	Response Set, $R = r_1, r_2, ..., r_{n-1}, r_n$
D	Plaintext Document Set, $D = d_1, d_2, ..., d_{m-1}, d_m$
P	Server Document Set, $P = p_1, p_2, ..., p_{m-1}, p_m$
D'	Known Document Set, $D' = d'_1, d'_2, ..., d'_{m-1}, d'_m$
X	Values in the Domain
ρ	Density of a database

Definitions

Term	Definition
Token-Query Pairs	Issued and plaintext variants of keyword
Setup Leakage	Information leaked at setup
Query Leakage	Information leaked when query is issued
Access Pattern	Set of document identifiers
Query Pattern	Ability to observe tokens
Rank	How many documents have the same value or lower
Document Size	Bit size of document
Response Volume	Number of documents returned
Partial Document Set	Set of server files in plaintext
Similar Document Set	Set of similar files to the server
Known Queries Set	Set of known token-query pairs
PQ-Tree	Structure to get order
Query Distribution	Distribution of how queries are issued

1 Introduction

Nowadays people upload substantial amounts of information to the cloud, such as Facebook, Google (Drive) or other cloud service providers (CSP). This information is at risk of being leaked as large data breaches happen quite regularly. The leaked data includes personally identifiable information (PII) such as phone numbers, addresses, and medical records [21, 46]. Data breaches are a societal problem because the gotten information could, for example, be used by scammers for password guessing. Leakage of PII can be avoided by using searchable symmetric encryption (SSE). Using SSE users keep search functionality, but CSPs are unaware of what is being searched or what is in the documents. Even though SSE schemes work and solve the problem partly, the schemes still leak a certain amount of non-negligible information. This information can be exploited. It is important to analyze the SSE leakage and see what attackers can learn from it. New attacks can be used to develop countermeasures that will increase the safety of the developed SSE schemes. It is thus crucial that new attacks are developed and tested.

1.1. Review Of Searchable Symmetric Encryption

Searchable symmetric encryption schemes were first taken developed by Song et al. in 2000 [48]. Following their scheme, more and more were designed. These new schemes have either become more secure or allow for a wider range of functionality such as range queries, Boolean queries or conjunctive queries [4, 23, 35, 53].

SSE schemes allow users to upload their information in an encrypted form at a CSP while retaining search functionality. During the setup, the user determines what keywords she wants to be able to query, encrypts these as an index, and sends this, together with the encrypted documents to the CSP.

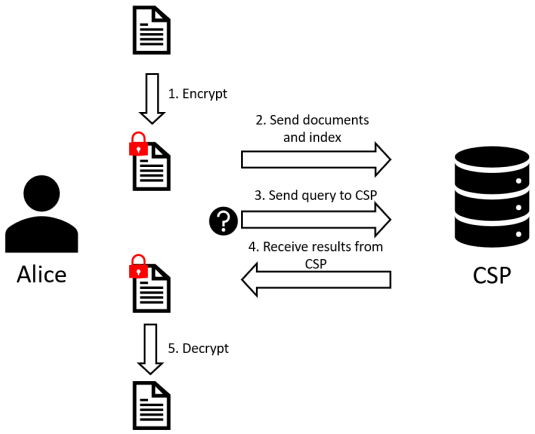


Figure 1.1: Basic SSE Scheme

1.1.1. Range Searchable Symmetric Encryption

Range queries are the sole focus of this thesis project. Range queries allow a user, Alice, to search for documents that match the range from a to b. Meaning that the value x of the returned document(s) has to be $a \leq x \leq b$. In our case the type of range databases will be one-dimensional, meaning that there will only be a single column that the user can query at a time. In multiple dimensions, a user could query multiple columns at the same time. Some schemes support two dimensions using smart solutions but that is out of the scope of this project. Range query systems are useful for, for example, medical databases where there are a lot of numerical values. Furthermore, range query systems are different from normal keyword SSE schemes, as documents can only have one value instead of multiple.

1.1.2. Query-Token Pattern

An user queries for a keyword/query using a token generated based on it. The server can use this received token to return the documents that match this token. It does this by using the token in a mathematical function. The server should not be able to learn what the plaintext query was for the created token nor what the document's plaintext values are. An illustration of this process can be found in Fig. 1.1. An attacker can observe this token and see if the same token is used more than once. The list of different observed tokens is called the *query pattern*. Furthermore, an attacker might already know what keyword/query matches with an observed token. This can be used to learn information about the database.

1.2. Leakage

Even though schemes have gotten more secure, different schemes leak specific types of information. This information can be divided into two main categories. *Setup leakage*and *query leakage*. Setup leakage leaks to the server or an adversary when the client sends the full encrypted database and encrypted keyword index to the server at the setup. This leakage, for example, contains the number of documents sent, and the domain size of the keywords. Query leakage occurs whenever a query is issued. This includes the *query pattern* or *access pattern*. The query pattern is the list of tokens. This leakage also allows an attacker to see when a user searches for the same query and how often they might search for it in comparison to other queries and use that. Looking at the access pattern, the list of documents returned, show if certain tokens respectively keywords have a lot of matching documents. If furthermore allows us to see when certain if two documents have overlapping tokens or keywords. These leakages will be explained in section 2.

1.3. Research Questions

New and secure SSE schemes are important to develop. They allow users to store their information more securely. To make such schemes secure, it is important to know what the leakage is and how the leakage can be exploited. This exploitation shows a need for mitigation techniques or more secure schemes. Lastly, different query distributions exist and affect the accuracy of attacks [18, 37]. These aspects result in the following main research question: **How can the leakage and structure of range searchable symmetric encryption schemes be exploited and what countermeasures work?**

This results in the following sub-research questions:

- **RQ1:** What are the leakage information of searchable symmetric encryption schemes supporting range queries?
- **RQ2:** How can current attacks on range and single keyword schemes be optimized?
- **RQ3:** In what context can auxiliary information be used?
- **RQ4:** How does the distribution of queries play a role?
- **RQ5:** What countermeasures could be used to evaluate the new attacks?

1.4. Contributions

The contributions from our research consist of multiple points, of which the most important one is a novel attack using a new direction which outperforms all current state-of-the-art attacks. This direction was found by looking at which attacks have been designed, what information they exploit and what kind

of performance they have. These attacks have been put in a diagram that can be seen in 1.2. A larger variant is visible in Fig. A.1 and a more elaborate variant is visible in Fig. A.2. Both are in appendix A. The figure shows that no previous attack has used known token-query pairs as the auxiliary input to their attack for range databases. Furthermore, there is only one access pattern-based attack that is agnostic and that works on sparse data sets. All the other agnostic access pattern-based attacks need a dense database to function. Furthermore, no attack has been able to handle more than one type of auxiliary information. To fill this gap, and to go in a new direction, we create an attack that can use known token-query data as a previously unused attack vector. Secondly, the attack can work on any query distribution and any density. Furthermore, this attack can handle multiple types of extra auxiliary information to increase the performance of the attack. Lastly, the developed attacks outperform all current state-of-the-art attacks that were tested.

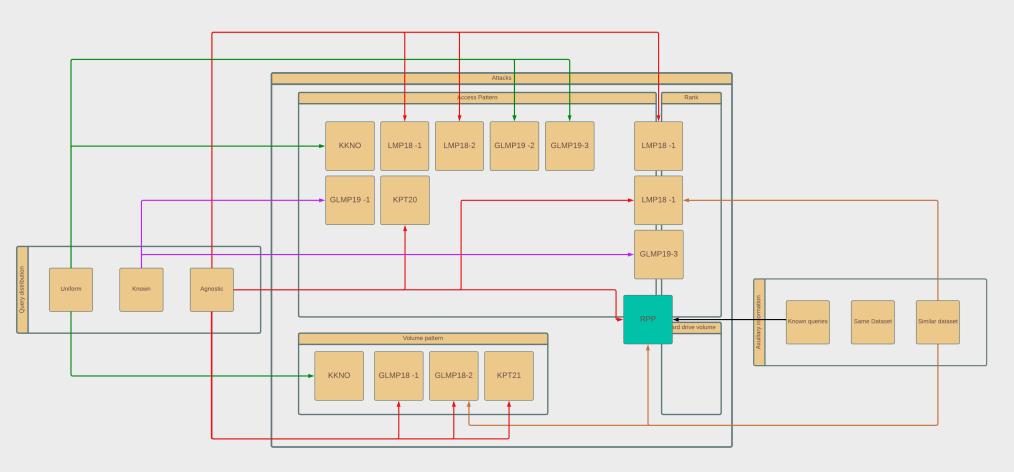


Figure 1.2: Basic SSE Scheme

The specific contributions of this research are listed below:

- New attacks to reconstruct queries and documents using PQ-Trees as a base which can be extended using rank or volume auxiliary information.
- Compared the developed attack to current state-of-the-art attacks
- Evaluated query distributions on the attacks.
- Comparison of several countermeasures of which ”Blocked Queries” severely impacts the performance and ”Wrap-Around Queries” negates most attacks.

1.5. Outline

The next chapter (chapter 2) goes into background knowledge. Chapter 3 will go into what others have done in this field of research and what other relevant options exist. The chapter regarding related work will also go into the current state-of-the-art attacks that the novel attack will be compared to. Furthermore, this chapter will also discuss the countermeasures used in this thesis. Chapter 4, Novel Attacks, will explain, in detail, the different novel attacks. Starting with the basic attack, the refined attack variant and lastly the extensions made using auxiliary information. Afterwards, we will show the results of the new attacks and the countermeasures. Next, we will discuss the interpretation of the results and the limitations of our work, after which we conclude this research thesis. Lastly, we look at what avenues can be explored still.

2

Background

2.1. Symmetric Searchable Encryption

The basis of SSE schemes that support range queries has been explained before. More details and the notations will now be discussed.

Firstly, queries are the plaintext form, while tokens are the encrypted variant of queries. $|A|$ indicates the size of set A. The plaintext database is defined as $\mathcal{D}$ and has document identifiers d_i where $i = [0, n-1]$ and $n = |\mathcal{D}|$. The server documents are denoted by P and server documents are denoted as p_i. In a range-query supporting SSE scheme, each document has one value in the domain $\mathcal{X}$ and the function $val(id)$ returns the corresponding value of the document identifier.

A user executes a query q_k out of all possible queries Q. These queries get transformed into tokens t_k out of all possible tokens T which the server then processes. The server returns a response out of all responses R. Here response r_k is the response to token t_k for query q_k. The observed token is called the *query pattern*. The query q_k consists of a range $[a, b]$ which is inclusive where $a \leq b$. The values for a and b are from the domain $\mathcal{X}$. This domain can be any numerical group. In our setup, $\mathcal{X}$ consists of integers $[1...L]$ where L is the maximum of the domain. r_k consists of a list of document identifiersp_i that have $a \leq val(p_i) \leq b$ for the corresponding $p_i \subseteq \mathcal{P}$. This list of document identifiers is also called the *access pattern*. The response length or response volume of the response, r_k, is $|r_k|$. Something unique to range SSE schemes is that they have a concept called *density*. This concept describes what percentage of values are represented by documents in the database. If the density is not 100 percent, then the database is not dense. If, for example, the density is 50 percent, then 50 percent of all the values in the domain are present in at least one document, while if the density is 100 percent then every value is represented by at least one document. The lower the density, the more sparse the dataset.

Lastly, there is an aspect called static and dynamic schemes. Static schemes are schemes that allow a user to only upload the document set and keyword index at the setup after which no alterations are possible. A dynamic scheme is a scheme that allows a user to change the database at run-time. These changes include but are not limited to updating documents by adding or removing keywords or documents and changing permission levels in schemes such as mPECK [31].

2.2. Leakage

As mentioned earlier, two types of aspects get leaked from SSE schemes, setup leakage, $\mathcal{L}_1$ and query leakage $\mathcal{L}_2$. Setup and query leakages have strict definitions that are dependent on the security level of a scheme. The security levels go from $L4$ to $L1$ as defined by Cash et al. [13].

2.2.1. Leakage Levels

Cash et al. were the first to formally define how much of $\mathcal{L}_1$ and $\mathcal{L}_2$ get leaked. They have four security levels, from $L4$ to $L1$, from largest to smallest [13] in the amount of information that gets leaked. If an attack works on a specific security level it will also work on a higher level, as that level leaks more

information. The different security levels and exactly what is leaked by them is described below.

The first leakage level $L4$ is defined as ”Full plaintext under deterministic word-substitution cipher” [13]. This means that every word in a document gets encrypted using deterministic encryption. This encryption allows anyone to count the number of identical keywords in their encrypted form, see how long the message is, and see the order of the keywords. This means that older and more basic dictionary attacks such as frequency analysis can be executed on $L4$ schemes. Furthermore, if a plaintext document is known, and can be matched to its encrypted form, then one knows which encryption is for which plaintext keyword. $L4$ is thus not considered sufficiently safe enough. Fig. 2.2a shows the repetitions and encryption of the keywords. One can also see how the keywords in documents are hit. The keyword gets converted into an encrypted form that is matched with the documents.

The $L3$ level leaks less and is defined as a ”Fully-revealed occurrence pattern with keyword order” [13]. $L3$ leaks the encryption of keywords in order of their appearance in the plaintext document. Yet, this time, any duplicate keywords will only be present a single time in their encrypted form. That duplicate keywords have been removed is visible in Fig. 2.2b. How the search tokens are handled is identical to $L4$. $L3$ is more secure, but as shown by Cash et al. and by our replication of their experiments, still leaks a considerable amount of information to an attacker.

For example, knowing some plaintext documents and their encrypted forms, allows an attacker to link several encryptions to the actual keyword. They then know a reasonable amount of keywords from other documents [13]. Fig. 2.1 shows the results from our reproduction of the attack by Cash et al. [13]. The reproduction was run on the Enron dataset [17] and contains 30109 files. The experiment shows that one only needs two documents to know up to 20 percent of the keywords for half the documents. If one knows as little as 20 documents out of the 30109, then an attacker knows 50 percent of the keywords for more than 80 percent of the documents. Secondly, if one would be able to inject documents in $L3$ schemes due to it being an active attacker, then knowing the encryption of all plaintext keywords is trivial [13]. This shows that an $L3$ scheme is still quite unsafe in many circumstances.

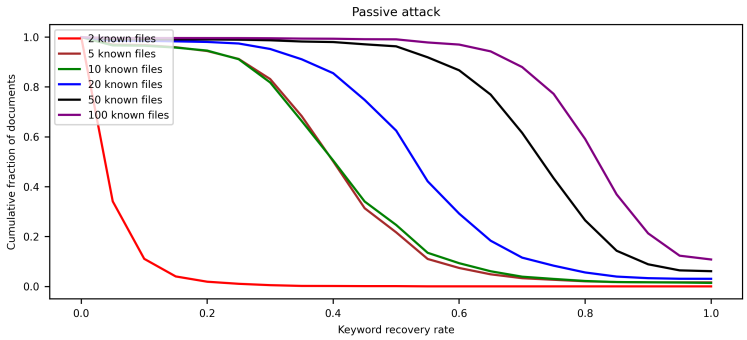


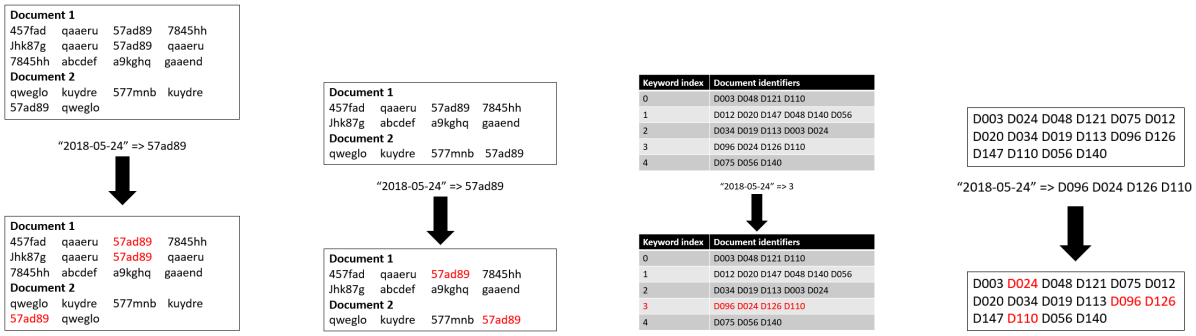
Figure 2.1: Keyword recovery rates for Enron for an L3 scheme. Re-implementation of the attack described by Cash et al [12]

The more secure level $L2$ was explained by Cash et al. as: ”Fully-revealed occurrence pattern”. This security level leaks which documents match which encrypted keyword. Yet, it does not show in which order the keywords were present in the documents. It does however leak the co-occurrence pattern of the tokens, the corresponding number of documents and how many documents match specific tokens. The co-occurrence matrix is a matrix of token x token or query x query. The values in the matrix are how often both tokens or queries appear together in the document set [18]. One can see $L2$ as an inverted index with the tokens as keys and the documents as the values. This can also be seen in Fig. 2.2c. The attack mentioned above would not be possible on $L2$ schemes. This is because the order of keywords is not present anymore.

Lastly, $L1$ leaks the least and is currently seen as the most secure. $L1$ is defined as: ”Query-revealed occurrence pattern” [13]. This means that it leaks what $L2$ leaks, but only for the queried keywords, meaning that one cannot compute a co-occurrence matrix in advance. A full co-occurrence matrix can only be made once all the possible queries have been observed. Attacks that work on an $L2$ scheme, will only work after all queries have been observed in the case of $L1$ schemes. That an attacker

only gets information one query at a time is visible in Fig. 2.2d. They only know that the token consisting of random symbols the server received for $2018-05-24$ matches with $D024, D096, D126, D110$ and nothing about the other documents and queries.

Our attacks mainly focus on L1-level schemes because we wanted an attack that can work independently of the leakage level.



(a) Search of keyword $2018-05-24$ on a $L4$ leakage scheme

(b) Search of keyword $2018-05-24$ on a $L3$ leakage scheme

(c) Search of keyword $2018-05-24$ on a $L2$ leakage scheme

(d) Search of keyword $2018-05-24$ on a $L1$ leakage scheme

Figure 2.2: The different leakage levels

2.3. Query Distribution

Query distribution has played a large role in the attacks on SSE. This is both for keyword and range SSE schemes. Damie et al. showed that the accuracy of their single keyword attack depends highly on the used distribution [18]. That distribution plays a major role was confirmed by Groot Roessing [25]. That this problem also exists in the domain of range queries, was pointed out by Kornaropoulos et al. [37]. A lot of prior attacks work either on the assumption that the distribution is uniform, or that the query distribution is known [37]. Yet the query distribution will not always be uniform. Users are more likely to search for shorter ranges in certain situations, such as when they know what they are looking for [37].

In the next section, we will explain the three distributions our research used in more detail. Two graphs will be shown per distribution and how exactly these distributions were generated will also be explained. We will follow the same principle as done by Kornaropoulos et al. [37]

2.3.1. Uniform Distribution

If the distribution of the range queries is uniform, then the queries $\mathcal{Q}_1 = [1, 80]$, and $\mathcal{Q}_2 = [5, 5]$ have an equal probability of being issued by a user. Certain attacks such as the KKNO [34], GeneralizedKKNO [27], and ApproxValue [27] attacks only work on this distribution and completely fail if a user uses a different distribution [37].

Fig. 2.3 shows that all the queries have an identical colour in the heat map, indicative of them all having the same probability and it being uniform. If one looks at Fig. 2.4, it is clear that the middle values of the domain have the highest probability to be present in a query, while the values at either end have a significantly lower chance of being present in a query. Relatively few queries encompass the value 1. Of such queries, there are only as many as the size of the domain. While exactly half the number of queries have the middle value in their range.

In the code, this uniform distribution was achieved by assigning the probability $1/|Q|$ to each query in the list of all the queries. This also ensures that the sum of all the probabilities is equal to 1. Lastly out of all these queries, a subset was chosen without repetition to ensure that unique queries were issued.

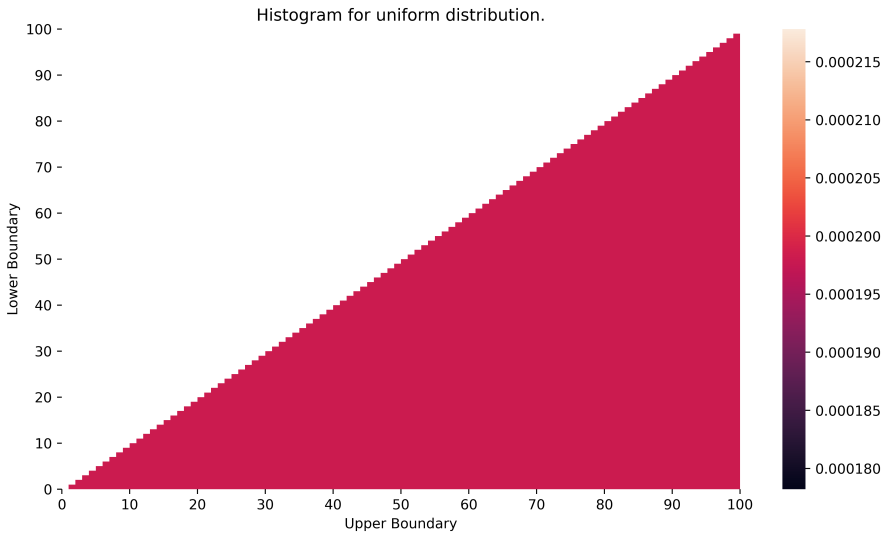


Figure 2.3: Heat-map for the uniform distribution

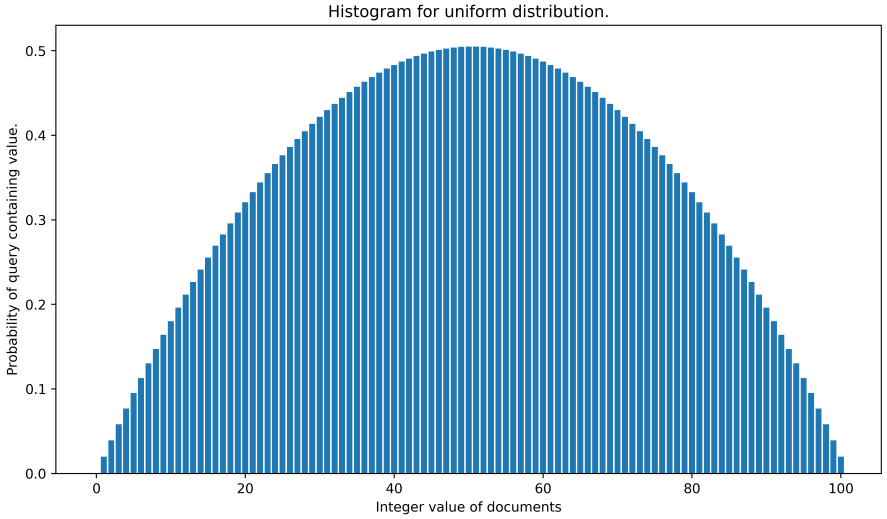


Figure 2.4: Histogram for the uniform distribution

2.3.2. Short Range Distribution

In the short-range distribution, queries which have a shorter range have a higher probability of being issued than queries that encompass a larger range. As mentioned, such a distribution might be more likely to occur than the uniform distribution. This distribution is also called the span distribution by Kornaropoulos et al [37].

To make it clear as to what this distribution means, we turn your attention to Fig. 2.5. There one can see that the diagonal line has the most distinct pink colour, the next line is a bit less distinct, and the next is even less distinct, etc. This happens until the probabilities of the queries happening become 0 and the colour is black. This shows that the short ranges have the highest probability of being chosen.

Fig.2.6 shows that most values have a more similar probability of being present in a query, than with the uniform distribution. Even though it has the same kind of curvature, the difference between the maximum and minimum probability for a query is 0.2 in the short-range distribution compared to 0.5 with the uniform distribution. Yet, even with this distribution, the endpoints are again less likely to be contained in the queries than the middle values are.

To get this kind of distribution we followed the same principles as Kornaropoulos et al. [37]. A beta distribution was created that generates a list of values that will be used for the different queries. The queries which have a width of length 1, get assigned the highest values as probabilities from the list of values after which the next set of values gets assigned to the range queries with a width of 2, etc. Afterwards, the probabilities of all the queries get normalized to ensure that the sum of the probabilities is equal to 1. We then select the necessary queries without repetition, thus if one requires or observes more queries, queries with a longer range will slowly start to turn up more and more. We only focus on the distribution of $(1, 3)$ for (α, β) in contrast to Kornaropoulos et al. who also did experiments with $(1, 5)$ and $(1, 20)$ $\alpha - \beta$ distributions [37]. We have deemed the other ranges out of scope for our setup. Because one short-range distribution is enough to show the difference compared to the uniform distributions. And as we are taking unique queries, making the short-range distribution more extreme does not significantly change the observed queries.

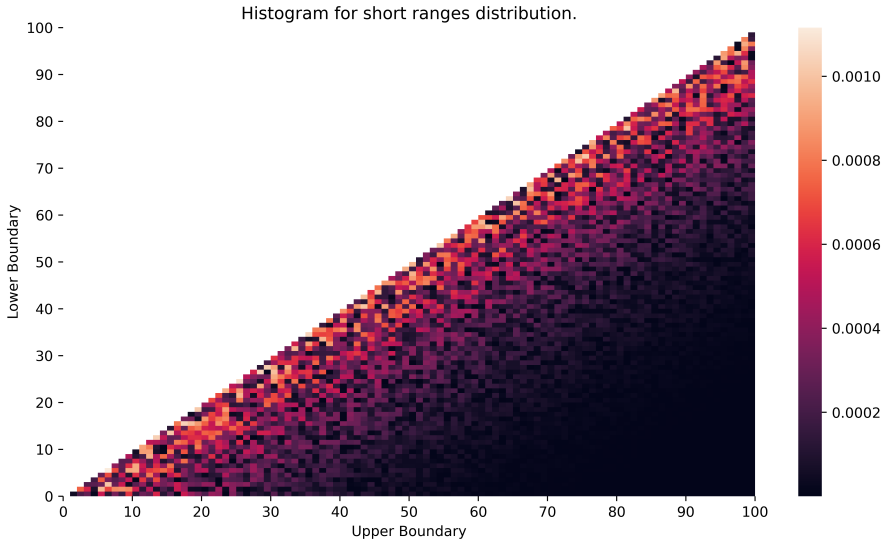


Figure 2.5: Heat-map for short-range query distribution

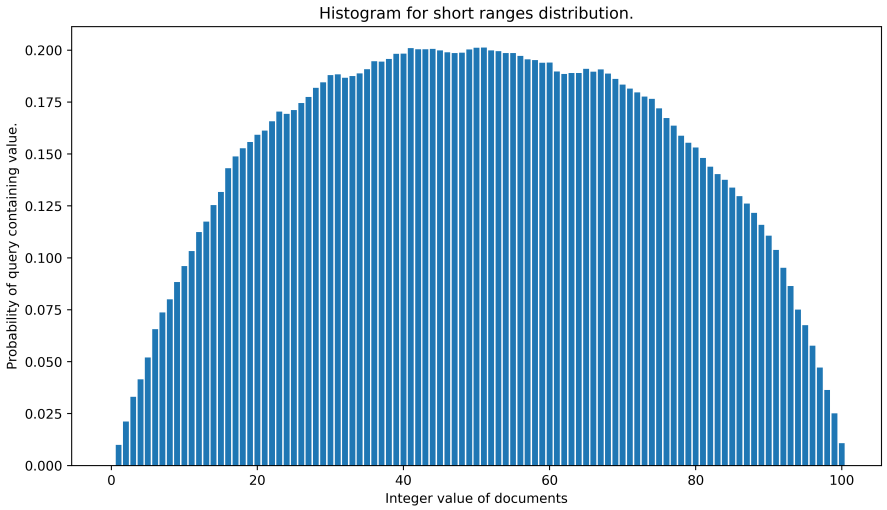


Figure 2.6: Histogram for short-range query distribution

2.3.3. Value Centred Distribution

The last distribution that was coined in ”The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution” is the value-centred distribution [37]. This distribution is a distri-

bution that assigns high probabilities to queries that contain specific values. This distribution simulates a user who wants to query for documents containing specific values and their surrounding values. This can for example be specific dates for people their birthdays or specific medical values for a medical professional.

Fig. 2.7, shows that there are several regions in the heat map that are more brightly coloured than others. These are the regions where the specifically chosen values were present in the queries. The chosen values for this specific graph were: $89, 53, 10, 59$. This means that queries that had 89 in their range were assigned the highest probabilities. This explains the change of more distinct colours after the 89 mark as the queries needed that as a minimum upper boundary. The vertical bar there can be explained too as the lower boundary can be anything from 1 up to 89 but not higher. The second group of somewhat bright blocks is visible on the left of the earlier group. This was from the chosen value of 53. As one can see, the colour is less bright as the assigned probability values are less per each chosen value. Here again, the same style shape occurs as before with the same reasoning.

The specific values chosen are also clearly visible in Fig. 2.8. One can see the dip at 89 after which the values do not show up as often anymore because there are fewer queries which have a high-end point that encompass all chosen values. Furthermore, one can also see the increase in probability until it reaches its highest value at 53. This is because a large number of queries which contain 89, also contain 53 and thus the 53 shows up in almost 80 percent of the queries.

This distribution was obtained using the functions described in [37]. A list of values was set up using a beta distribution with 1 and 3 for α and β respectively. The largest values in this list were assigned to the queries that contained the first randomly selected value. Then the next group of values was assigned to queries that contained the new randomly chosen value. We kept selecting random values and assigning probability values to the appropriate queries until there were no more values in the earlier gotten list. The sum of all probabilities was then normalized to get 1 again. In [37], they have two more value-centred distributions that we skip because one value-centred distribution would already tell us a substantial amount about how such a distribution impacts performance.

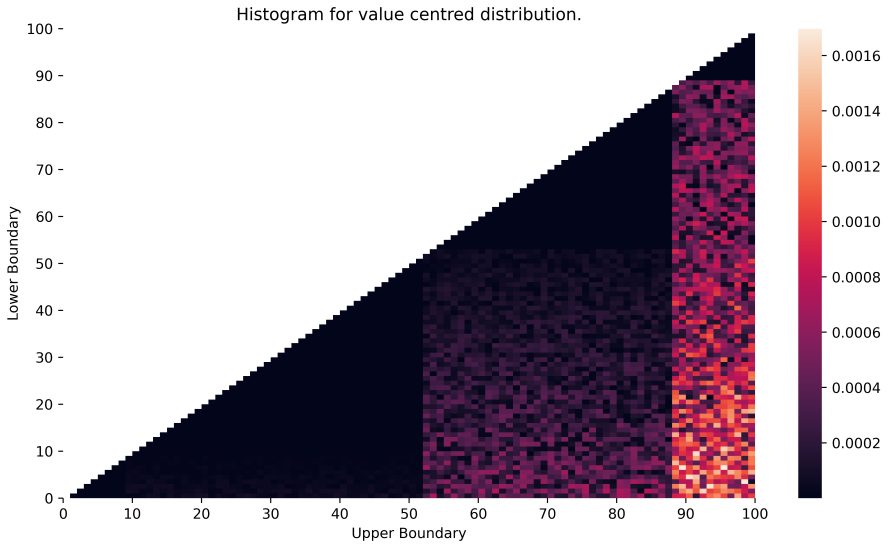


Figure 2.7: Heat-map of the value-centred distribution for the values $89, 53, 10, 59$

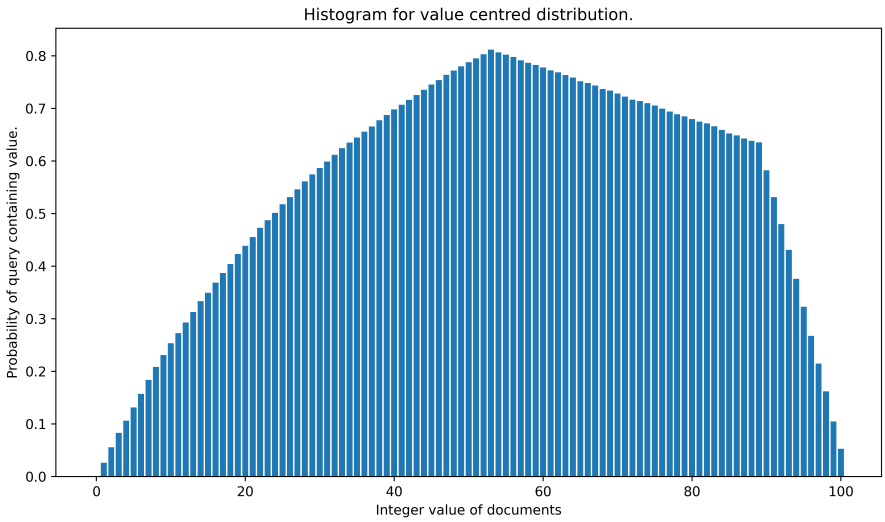


Figure 2.8: Histogram of the value-centred distribution for the values $89, 53, 10, 59$

2.4. Attacks

In this section, we will look at the types of attacks that exist, what can be achieved by such attacks and what information can be available to an attacker to perform their attacks.

2.4.1. Attack Types

First off, there are two main types of attacks, these are active and passive attacks. In an active attack, an attacker actively does something with the database or with one of the SSE protocols to learn more information. An active attacker can, for example, inject files into a server [55]. A passive attacker is an attacker that takes an observing role. He or she does nothing other than observe network traffic or see which documents are returned. This also means that for a passive attacker fewer assumptions are required and the attacker is more likely to exist on the server or as a CSP who is honest but curious. Such a CSP would then do all it can within the limits of the protocol to get the most knowledge possible.

In the range SSE field there are two targets for attackers. Either query reconstruction or value reconstruction. If one can do the second, then an attacker can also make a list of possible queries for observed tokens. While if one is only able to match queries to tokens, then they are not necessarily able to decipher the exact values associated with documents, but they can have an idea of the possible values for the documents matching a query. Our attacks can do both reconstructions very well.

2.4.2. Auxiliary Information

An attacker might have been able to gather extra information that it can use for its attack. This information could have been gathered due to previous data leaks or by scanning the internet for documents that are in the same field. It is even possible for an attacker to have obtained unencrypted documents of the user on the CSP before they started using the encrypted plan or program. In our setup, the three auxiliary data sets that are relevant are the partial document set, similar document set, and known tokens-queries.

Partial Document Set

The partial document set consists of parts of the encrypted dataset that gets sent to the CSP. If the attacker has a complete partial dataset, then they know all the information of the documents on the server. A partial dataset can be used to learn what query matches with which token for example, or which encrypted documents correspond to which value. The fewer documents are in the partial dataset, the less similar the auxiliary dataset is to the original dataset. This similarity can be defined as the distance between the histograms of both distributions of the database. The closer it is, the better the dataset can be used because it reflects the actual distribution more. Having fewer documents often makes an attacker less accurate, but an attacker is more likely to own fewer documents. Even having

a small amount of partial information means that the attacker needs to have been able to gain access to parts of the system or data somehow. This can be considered quite a strong pre-condition for an attack using this information to even start.

Similar Document Set
A similar document set is a dataset that is more likely to have been obtained by an attacker. Such a dataset is a dataset that has a similar distribution to the dataset of the user. Attacks using a similar document set can have a high accuracy [18] and they are more easily attainable, making it more likely for an attacker to be in the possession of such a dataset. As stated in [28, 27] an attacker can use data that was released in previous years or data from a similar region that was made public. An attacker would then not even need to perform malicious acts to obtain the data.

Known Queries
The last auxiliary information that is relevant to our attack is the known queries. Known queries are query-token pairs that the attacker has in their possession. These can either be chosen by the attacker or not. If the attacker can choose them, then he or she can choose the queries that benefit them the most. This was also shown by Damie et al. [18]. Knowing queries allows an attacker to set certain aspects and knowledge. For example, a sub-matrix of co-occurrences such as in [18]. There is some debate in the academic world as to how realistic it is to have obtained such queries, but we follow the reasoning of Damie et al. who state that an attacker could do a file injection attack to recover several known query-token pairs [18].

Ranks
An attacker can use the partial or similar dataset to get for example rank information. Rank information can be used to define values in documents based on the ranks. We use the same definition of ranks as Lacharite et al. who state that the rank of a value z "is the number of records having a value less than or equal to z" [38].

2.5. PQ-Trees

The designed attack assigns values to documents based on the order of those documents. The order of those documents is determined using a structure called PQ-Trees. This is aptly named as the structure has P and Q nodes in a tree. The idea of these PQ-trees was conceived by Booth and Lueker in 1976 [6]. In this section, we will explain what a PQ-tree is, what the different node types are, what a reduce operation means and give an example of a PQ-tree to visualise it.

What are PQ-Trees As defined by Booth and Lueker, a PQ-tree "represents a class of permissible permutations" [6]. The PQ-tree is a structure that, given enough input, allows one to find the possible permutations for the initial input. One has to first input all the identifiers to the PQ-tree after which it can do reducing operations. At the end of all these reduce operations, one can figure out the list of possible permutations, based on if it is a Q or P node. Each of these nodes has leaves which then contain the identifiers. There is no limit on the number of leaves a node can have [6].

Node Types As mentioned there are two types of nodes, these are P and Q nodes. A P node, depicted as a round node in this thesis, is a node whose leaves can be ordered in any order [6]. If for example a P node has the leafs a, b, c, then the possible permutations can be $(a, b, c,, a, c, b, b, a, c, b, c, a, c, b, aandc, a, b)$. A Q node, which is visualized as a square node in this thesis, and which has the same leaves only allows for the following options: (a, b, c, c, b, a). It has been determined that b has to be between the other two points. For the Q node, only the order as shown in the node or its inverse is allowed. Q nodes thus already tell us more about the order of its leaves. The identifiers get placed in proper nodes through a function called reduce [6].

Reduce The function reduce has been coined reduce "because it reduces the possible class of permutations to only those which satisfy the constraints." [6]. The constraints mentioned, are the constraints of order that have been previously obtained. If there has been an input a, b, c and one gets the input a, b. then one knows that c is either before or after a, b. It can not be in between. This operation can

be done for all inputs. If it ever happens that the reduction cannot be done because it does not satisfy the constraints, then there has been something wrong with the input and the algorithm notifies the user [6]. Through these reduce operations, the order of the document identifiers in the database can be determined. A small example is given below.

Example One can see the visualisations of this example in Fig. 2.9 until Fig. 2.16. First, all the document identifiers are put into the PQ-Tree. This results in a P node at the top and as leaves the document identifiers as there is no order yet. This can be seen in Fig 2.9. We input our first observed access pattern with document identifiers $2, 5, 8$ marked in red. It is then known that these belong together but it is not yet known in which order, thus they are put in a P node. These steps can be seen in figures 2.10, 2.11 and 2.12. The next access pattern, $2, 3, 6$, gets used as input for the next reduce. This means that the document identifier 2 is between $3, 6$ and $5, 8$ which means that these get reduced into a Q node with the groups as P nodes under the Q node. The result of the operation is visible in the next figures. In the last figure, the final result of all the reduce operations is visible. The resulting tree allows us to know the relative ordering of the documents. The ordering is up to reflection. This reflection is something that can be broken using the developed algorithm later on.

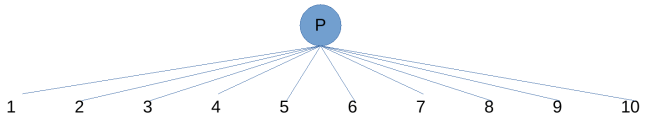


Figure 2.9: PQ-Tree step 1

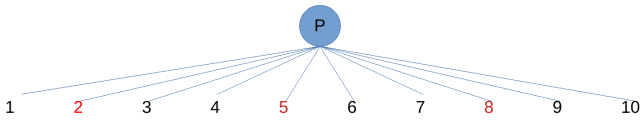


Figure 2.10: PQ-Tree step 2

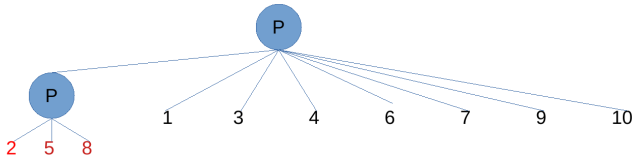


Figure 2.11: PQ-Tree step 3

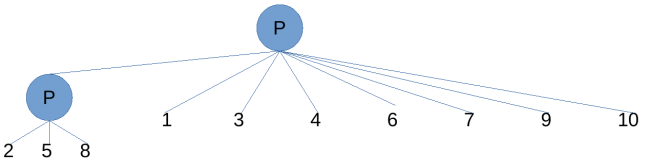


Figure 2.12: PQ-Tree step 4

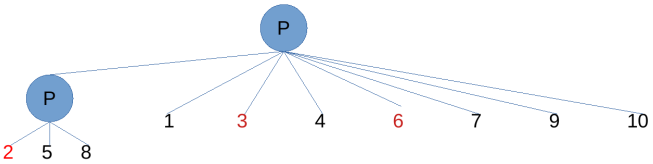


Figure 2.13: PQ-Tree step 5

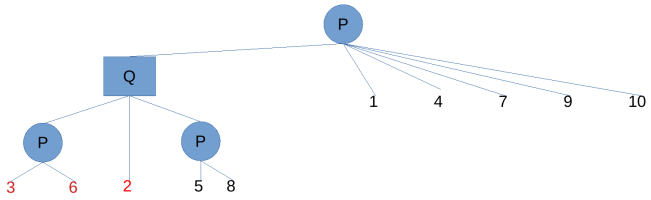


Figure 2.14: PQ-Tree step 6

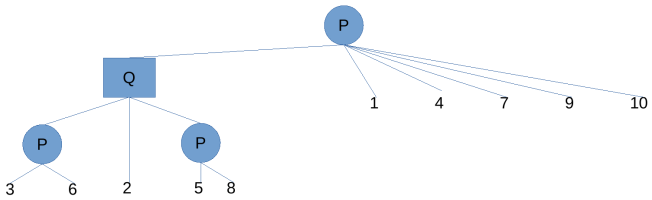


Figure 2.15: PQ-Tree step 7

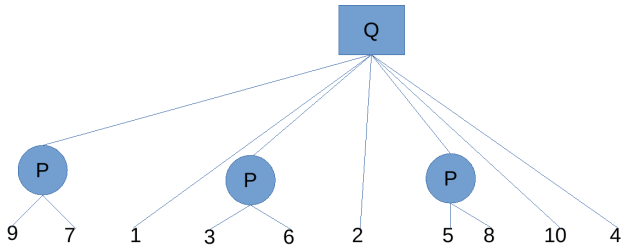


Figure 2.16: PQ-Tree step 8

2.6. Attack Overview

Attack	Auxiliary Information Used	Leakage	Query Distribution	Passive Attack
Basic RPP-Attack	Known Token-Query Pairs	Access Pattern	Agnostic	✓
Refined RPP-Attack	Known Token-Query Pairs	Access Pattern	Agnostic	✓
Rank RPP-Attack	Known Token-Query Pairs Similar Dataset	Access Pattern Rank Information	Agnostic	✓
Document Size RPP-Attack	Known Token-Query Pairs Partial Dataset	Access Pattern Hard Drive Information	Agnostic	✓
GLMP18 [28]	Similar Dataset	Response Volume	Agnostic	✓
GLMP19 [27]	Similar Dataset	Access Pattern Rank Information	Known	✓
KPT20 [37]		Access Pattern Query Pattern	Agnostic	✓

Table 2.1: Overview of the attacks, the information they employ and their leakage.

Table 2.1 shows some of the attacks and the knowledge and leakage needed for them to be executed properly. These attacks will be discussed in more detail in chapters 3 and 4. As cone can see, almost all attacks employ the access pattern and are agnostic. Furthermore, they are all passive attacks.

3

Related Work

The field of SSE is not a new one. As mentioned, the first SSE scheme was defined by Song. et al. [48] in 2000. These schemes supported very basic functionality such as searching for single keywords and were not always as secure as one might want. As time went on, more research was done into SSE. In this research field, there are three directions, 1. the security of the schemes, 2. the functionality of the schemes, such as what type of searches and the efficiency of schemes and 3. the attacks on the schemes. In this chapter, the three directions will be discussed after which previous work that has defined the direction we have taken will be mentioned. Furthermore, in this chapter, some countermeasures and how they work will be explained in detail.

3.1. Schemes

The first scheme [48], was very basic and has been the groundwork for many others. Nowadays schemes can do a lot more due to developments in research. One such option is conjunctive keyword searches. In the beginning, when a user wanted to do a conjunctive keyword search, the user had to send multiple single keyword searches or the server itself had to break up the query into multiple parts. Here the problem lies that the server learns the access pattern for each keyword individually. Such a scheme would not only have a longer run-time, but it would also leak more than is necessary. An attacker would be able to do single-keyword attacks to determine the encrypted conjunctive query. To counter this, Cash et al. [12] developed a scheme that can efficiently handle large document sets in a conjunctive keyword setting. Several other conjunctive schemes use this as a basis for their scheme. The scheme is called Oblivious Cross-Tags (OXT). OXT is a scheme in which the search complexity scales with the number of documents matching the estimated least frequent keyword in the conjunction and not in the number of documents in the database. The scheme also allows for Boolean queries to be done on the database in the same efficient manner [12]. Other conjunctive schemes can be found in [30, 40, 20].

Other than Booleans and conjunctive queries, nowadays wild card query schemes are also possible. A wildcard query is a query where a user can insert an $*$ as a wildcard symbol. The user can then send the query ”f*r” and get documents which contain keywords such as ”far” and ”for”. The scheme matches any keyword as long as it starts ”f”, can have any character in the place of $*$ and finishes with the letter ”r”. One such scheme that has this functionality is researched by Sedghi et al. [47], who created a hidden vector encryption (HVE) scheme. This scheme is ”based on bilinear groups of prime order, which supports vectors over any alphabet” [47]. Further examples of wildcard schemes are: [8, 54].

All these schemes have one user in mind. Yet, one can imagine a system where multiple users need to be able to upload documents and search for documents. Imagine, for example, a hospital in which both the doctor and the nurse need to be able to search through a specific set of documents. Or if a nurse uploads a document, only the appropriate doctors should be able to also search it. A scheme that has these characteristics was developed by Hwang and Lee [31] and is called mPECK. The definition of mPECK is ”multi-user public key encryption with conjunctive keyword search” [31]. This scheme allows a user to designate specific users that are allowed to also search and decrypt the

documents using a form of public key encryption. They furthermore support conjunctive queries in this scheme [31]. A thorough analysis of multi-user schemes has been done by Bösch et al. [7]. We thus refer the readers to the paper by Bösch et al. [7] for more information on multi-user schemes.

3.2. Security

Security of SSE schemes is important as attackers or CSPs should not learn anything, yet if they do learn something it should be as little as possible. For this purpose, multiple defence and security measurements have been developed.

One of the defence mechanisms that one could use was developed by Goldreich and Ostrovsky in 1996 and is called Oblivious RAM (ORAM) [24]. They describe a solution called ”Square Root” ORAM [24]. This scheme allows for a situation where an ”adversary can observe the physical storage locations accessed, but the ORAM algorithm ensures that the adversary has a negligible probability of learning anything about the true (logical) access pattern” [49]. This means that the observer can see where the data is being read, but the actual hardware location of the value could be different. ORAM can be used in combination with SSE schemes but has a significant overhead in time complexity due to reading times, furthermore the bandwidth increases due to the setup. This is why its use is not yet widespread. More efficient solutions exist, such as Path ORAM [49] which uses a tree-like structure with buckets and a small client-side storage location to efficiently search. This structure is better but still not efficient enough from a performance standpoint as the performance of the Path ORAM is dependent on the number of buckets and the number of items each bucket can have. As time goes on more and more ORAM solutions are developed and they might one day be a logical countermeasure which does not incur too much of a performance hit.

In 2016 Zhang et al. [55] published a paper on file injections and showed that they, as long as they can inject the files, can achieve 100 percent accuracy in determining what query matches the observed token. Such an attack is detrimental to the security of users as an attacker could know everything. In their paper, they state that forward security is an avenue that needs to be explored. A forward scheme is a scheme that makes it difficult for file injection attacks to be performed properly. It ensures that documents added after a query has been issued and run, cannot be returned by that same query. This means that an attacker cannot replay tokens to get the newly added results but has to observe a new query which makes it difficult to match it against the earlier query [9]. Forward secure schemes have been designed by the scientific community and a well-known one has been developed by Bost et al. [9]. If the reader is interested in more forward schemes, other options can be found in [2, 14, 10]

While forward security is for adding documents, backward security is a security measure for the removal of documents. Bost et al. [11] were the first to properly look at backward security. They coined three different levels of privacy for backward security. These are in increasing order of leakage: type 1 - ”Backward privacy with insertion pattern”, type 2 - ”Backward privacy with update pattern” and type 3 - ”Weak backward privacy” [11]. Type 1 leaks the documents that are currently in the database that match the searched keyword and when the corresponding files were added to the database. Type 2 leaks the same as type 1 with the addition of stating when updates happened on the documents containing the searched keywords. These updates can be insertions of documents, deletions of documents or changes to what keywords are in a document. Lastly, type 3 leaks the same as type 2 plus the timestamp of a deletion combined with the timestamp of the insertions [11]. Backward security is a security feature that blocks an attacker from being able to retrieve files that were on the server before they gain access, thus limiting the amount of knowledge one can obtain. Other backward secure schemes that use the ideas of Bost et al. [11] as a basis are: [3, 15, 45, 50]

There are several countermeasures such as padding [13], obfuscation [16] and blocked queries [42]. We will discuss a number of these countermeasures in section 3.5 and show their performance on the different attacks.

3.3. Attacks

This section will discuss several attacks that have been developed, on both single and range keyword schemes.

The first attack that set the foundation for later attacks is published by Islam et al., called the IKK attack [32]. It was the first major attack on SSE schemes and showed that there was a need for more research into attacks and the security of schemes. The authors used simulated annealing to determine the best match for tokens and queries. This attack used a similar dataset as auxiliary information and constructed co-occurrence matrices. The attack had a high accuracy as long as a substantial amount of information is known to the attacker [32]. In our reproduction of their tests, we encountered that the attack, due to the simulated annealing aspect, can run for a long time before finishing. An improvement on the attack has been made by Groot Roessink et al. who added a deterministic aspect to it [26]. Furthermore, fully replicating it was rather difficult as the attack is parameter-dependent.

An improvement in attack performance was made by Cash et al. [13] who match queries based on their unique response lengths. They then use these matched queries to refine possible options for the remaining tokens and select the single remaining query for an observed token after filtering others out. They also have an option for when there are no unique response lengths but this variant incurs a large computational overhead as it goes past every starting combination of token and query [13].

A different attack that uses query leakage is explained by Liu et al. [41]. They create information using Google trends to see how often certain keywords were searched on Google. They then match this data to the search frequencies of the observed keywords using the query leakage [41]. This works quite well as long as google trends is an accurate representation of what gets searched by a large number of individuals. Oya and Kerschbaum [44] have improved on this using other leakage aspects.

Looking at attacks on range queries, the paper ”Generic attacks on secure outsourced databases” often called the KKNO attack [34] was the first one to break the confidentiality of documents in range databases. They were able to create two attacks, one using the access pattern and one using the response volume. The results of the attack were good. It was able to reconstruct all documents, yet at a large overhead cost in the number of queries that need to be observed and the run-time of the algorithm. They also state that for large data sets with more than 150 documents, the factorization method becomes impractical due to the run-time [34].

Grubbs et al. [28] approached the problem from a different angle. They observe the response volume of queries and use this to find cliques of elementary volumes. They then get the document counts for each value in the domain. They show that this can then be used to match queries to tokens as long as the different tokens have a unique response length. If this is not the case, the accuracy decreases quickly. A large advantage of this attack compared to the KKNO attack is that it requires a significantly lower amount of queries to be observed. The only disadvantage is that this attack requires an auxiliary dataset in the case of sparse data sets, to know which values in the domain are present. This is because it is not able to know where the gaps are [28].

The above attacks are attacks that only work on one-dimensional range query databases, however, range SSE attacks that work on multiple dimensions also exist. The first one was developed by Falzon et al [22]. The attack they have developed requires either the search pattern leakage or knowledge of which query distribution is being used. It furthermore requires all queries to have been issued at least once. Another paper that looked into two-dimensional attacks is [43].

3.4. Previous Attacks

Now that previous works have been discussed, we will now go into more detail about the papers that were built upon or that the designed attack will be compared against. These are:

- A highly accurate query-recovery attack against searchable encryption using non-indexed documents [18]
- The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution [37]
- Pump up the volume: Practical database reconstruction from volume leakage on range queries [28]
- Learning to reconstruct: Statistical learning theory and encrypted database attacks [27]
- Revisiting Leakage Abuse Attacks [5]

3.4.1. A Highly Accurate Query-Recovery Attack Against Searchable Encryption Using Non-Indexed Documents

The paper ”A highly accurate query-recovery attack against searchable encryption using non-indexed documents” [18] was published by Damie et al. in 2020. In the paper, they discuss an attack called the ”score attack”. This notation will be used throughout the paper too. The score attack is an attack that is used to recover the matches between queries and tokens with high accuracy. It is also one of the first attacks to properly show the power of using a similar dataset versus a partial dataset. They argue that a similar dataset is more easily obtained than a partial dataset [18].

The authors define two attacks. Their base score attack and their refined score attack. Damie et al. use the known query-token matches to set up a known query-to-query co-occurrence matrix. From this co-occurrence matrix, a similarity score can be calculated. If the chosen query for an observed token is the correct one, then one would assume that the co-occurrences of the token with other tokens are similar to the co-occurrences of the correct query with other queries. The closer the similarity of the two co-occurrences, the higher the score. Then for each token they observe, they choose the token which has the highest similarity score and choose that one as the final match [18]. Having known queries is something that our novel attack also uses. The exact implementation of this attack is explained in chapter 4.

In their refined version, they add a variable called ”refine speed”. This variable determines how quickly the algorithm finishes and how strongly it refines. The refining happens by selecting the top ”refine speed” matches that they are most sure of and setting these as ”known” values in the co-occurrence matrix. Then the algorithm goes again and checks for the remaining tokens. This allows for a stronger comparison of the remaining tokens as there is more to compare to. Setting the ”refine speed” to a low number allows for a lot of the tokens to have a very thorough comparison with the disadvantage of having a higher run-time. If the refine variable gets a higher value, the run time goes down, but so does the accuracy [18]. We have implemented this attack too and came to the same conclusions.

Lastly, they also tested two countermeasures. The countermeasures were proven to be effective and even completely nullified the attack if the keyword space became too large [18]. Showing the importance of looking at countermeasures and seeing which ones work and which ones do not.

3.4.2. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution

The first attack we will compare our attack to is described by Kornaropoulos et al. in ”The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution” [37]. They describe an attack called the ”Agnostic” attack, also called KPT20 in this thesis since it is agnostic to any query distribution used and it need not know the distribution beforehand [37]. The authors show the power of their attack on different types of distributions [37]. These different distributions are also used in this thesis report in the experiments to validate the performance of our attack.

KPT20 works by grouping all queries which return the same access pattern. The next step is to estimate the actual number of queries that should have returned the observed access pattern using an

estimator. A weight also gets coupled to each observed access pattern by looking at total the number of queries that returned the access pattern. This is because the more unique queries one has observed, the more sure one can be. Next, the weights and the estimated number of documents get put into a system of linear equations and solved. The following step is to normalize the approximated lengths. Once that has been obtained, one can assign values to documents based on the approximated lengths. For further info see the original paper [37].

The authors run experiments on their attacks and use the mean square error (MSE) and the mean of absolute error (MAE) as metrics for the attack. The described error is the difference between the actual value and the estimated value. Using the MSE results in a worse score if large differences occur [37]. The replication of this attack was written by Kamara et al. in the Leaker framework [33].

3.4.3. Pump up the Volume: Practical Database Reconstruction from Volume Leakage On Range Queries

Grubbs et al. published a paper in 2018 that focuses on observing the response length of tokens and uses those to get the document counts of individual values in the domain [28]. This works well on dense databases. On sparse databases, it is unable to find the exact document count per value as it does not know where the sparse gaps are. Yet our focus lies on a smaller section of this paper that introduces an attack called the ”CDF Matching attack” [28] which will be called GLMP18 from here on out.

The GLMP18 attack is an attack that focuses on linking queries to tokens, rather than focusing on database value reconstruction. The attack can be run as soon as one token is observed and thus does not require multiple tokens to be effective. The attack requires knowing the domain size and database distribution. This database distribution does not need to be the actual distribution making this a similar document type of attack. The attack works by first calculating a precision based on the confidence parameter and the number of documents on the server. The confidence parameter can be tweaked to lower the number of options per token while having a high chance of keeping the correct query in the list of possible options. For every observed token, it tries to match it to a list of possible queries. The expected fraction of documents a possible query returns are calculated and compared to the observed fraction of documents that was observed. If the comparison value is close enough to the precision value calculated earlier, it gets added to the list of options.

The authors report the results in their paper. This attack works reasonably well. Yet it is not able to guarantee that the right query is always in the list of options. This can result in quite a lot of false positives if the attacker assumes correctness. Furthermore, sometimes a lot if not all of the possible queries are added to the list. For an attacker, this can mean that it still does not learn as much as it wanted to. This is the second attack that the novel attack will get compared to. Similar metrics as described in this paper will be used in the thesis report.

3.4.4. Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks

Grubs et al. wrote another paper called ”Learning to reconstruct: Statistical learning theory and encrypted database attacks”. This paper introduces several new attacks based on the access pattern. They introduce 3 attacks. The first one is called ”GeneralizedKKNO” [27] but is not relevant to our research. The next attack they introduce is ”sacrificial ϵ-approximate order reconstruction” [27] also known as the sacrificial ϵ-AOR [27]. This attack uses the PQ-Trees explained earlier to get the order of the documents. Yet due to the sacrificial nature of the attack, if not enough queries are observed, it has to sacrifice documents which have values at either end of the domain. It furthermore requires that at least some records exist within a specific range [27] which means that this attack has some assumptions to be successful.

Lastly, they have created a value reconstruction attack [27]. This is an attack that uses the order output from the attack described above as input. It then returns the document values of the documents it has assigned values to. Here too it suffers from having to accept an approximate reconstruction because it does not have perfect knowledge. The attack has an auxiliary distribution of the database. This can again be a similar database instead of a partial one [27].

The attack works by first orienting the order in the right direction using a heuristic. Next, they calculate the number of records that were sacrificed. This is used to get the starting rank for the first group. From there they can calculate the ranks of the remaining groups. The ranks of these groups are used to get the actual value of documents in the groups. This value is determined using the auxiliary database and choosing the median value of the documents that are within the ranks [27]. This is the last attack used for the comparisons of the attack performance. The attack will be called the GLMP19 attack throughout the rest of the thesis report.

3.4.5. Revisiting Leakage Abuse Attacks

Blackstone et al. wrote the paper ”Revisiting Leakage Abuse Attacks”. This paper introduces several new passive and active attacks using partial information as their only auxiliary dataset. Most of their attacks are volumetric meaning that they look at the size of the documents on the hard drive in bytes. They further also introduce an attack concept called the subgraph attacks. This style of attacks uses graphs to link possible queries to tokens. The attacks described by them perform well as long as the chosen keywords have high selectivity, meaning that they are present in a lot of documents. If there are keywords with low selectivity the accuracy decreases quickly [5]. This was also confirmed in our experiments which are visible in Fig. 3.1b and Fig. 3.1a. The results show that it is important to analyze the query distribution. An attack can have a very high accuracy such as with the subgraph attacks for high keywords on almost any amount of partial knowledge. Yet if the keyword distribution is different the attack does not get above 25 percent accuracy if all documents are known. Significantly lowering the effectiveness of the attack in certain scenarios.

(a) Low Selectivity (higher score is better)

(b) High Selectivity (higher score is better)

Figure 3.1: Performance of the reproduction of the attacks in [5] on different selectivity

They have furthermore developed two volumetric attacks called the ”volume analysis” (VA) and the ”selective volume analysis”(SVA) [5]. SVA is an attack that scores higher. The first attack works by analyzing the volume of the documents on the server and the volume of the documents the attacker has at its disposal. The attacker, for every token it observes, sums up the volumes of the returned documents. It then maps it to the keyword that has the closest similar volume in the known partial dataset. As mentioned in the paper, this first attack does not take into account how much the attacker knows about the database. SVA does take this into account.

The SVA attack first looks for possible matches to a token by checking which queries have a volume that is equal to or less than the volume of the server documents, but higher than a specific minimum. It then further refines this selection by estimating the selectivity of the keywords. The selectivity here is the number of documents expected to have seen returned based on what is known. This selectivity is checked against the observed documents and if it does not meet certain criteria, then the keyword gets removed from the list of options. Once this is done for all possible keywords, the first one in the list gets selected as a match [5].

Lambregts et al. also used the volume of documents to match extra documents to increase the performance of the LEAP attack [39]. This avenue seemed very interesting to further explore and thus some of our auxiliary attacks use the document volume to link values. Exactly how is explained in the next chapter.

3.5. Countermeasures

In the past years, several countermeasures have been developed. Most of these countermeasures have been designed to work on single keyword schemes while only a few are available for range query schemes. We will focus on 2 countermeasures for range query schemes. Furthermore, a countermeasure focused on document volumes has also been looked at.

3.5.1. Blocked Queries

The first countermeasure used is called ”Blocked queries” and is developed by Markatou and Tamassia [42]. The countermeasure effectively adds extra documents to a query by increasing the range of the query that gets sent to the server. This increase in the range is handled, in their setup, by the program on the client side. This program also filters out the extra padded documents, and false positives. The trade-off here is a communicational overhead due to the increase in the number of documents that have to be sent over the network. The countermeasure is based on a security parameter called k. The higher the k, the more secure it is, but also the more overhead there is present. Markatou and Tamassia describe their countermeasure as follows, the client wants to send the query $[a, b]$ to the server. The countermeasures program then modifies the query using k to get $[k \cdot \lfloor a/k \rfloor, k \cdot \lceil (b+1)/k \rceil - 1]$. This means that a query $Q = [11, 15]$ using $k = 4$ would get modified into $[8, 15]$ which increases the range [42]. This means that there are fewer unique queries.

3.5.2. Wrap-Around Queries

The second countermeasure is once again created by Markatou and Tamassia and called ”Wrap-around queries” [42]. According to them, this countermeasure is more secure when the attacker knows the query distribution but it also comes with its downsides like an even larger communicational overhead. The countermeasure works in multiple steps [42]:

- Create a list of wrap-around queries that loop from the end of the domain to the front values, thus wrapping around and putting these in a buffer.
- Combine issued queries with wrap-around queries using a random coin.
- Split these up into singleton queries and shuffle them.
- Send these singleton queries to the server and have the client program filter out the necessary items.

Such a setup results in two large advantages, first it ”removes asymmetries due to the client’s query distribution” [42] and secondly, ”it turns the database into a cyclic buffer in the eyes of any adversary” [42]. The first advantage is that every value has an equal probability to be hit by a query. The second advantage follows from the singleton queries being sent. Our attacks using the PQ-tree would fail because there is no order to be retrieved. This is because there is no overlap between tokens in their range. Our attack could only work if we are somehow able to have single-range known token-query pairs. Then values could be assigned to the observed access pattern and create an order that way. We have decided to not look at this specific case as we deemed it too specific. For more information on the countermeasures, we refer the reader to the original paper [42].

3.5.3. Document Volume Hiding

The last countermeasure is the hiding of document volumes. This means there will be a countermeasure in place which ensures that all the hard drive volumes of documents are indistinguishable by making them all the same or equal to the closest multiple of k for a security parameter k. The higher the k the more security is available. Yet this comes, again as in other countermeasures at a cost of communicational overhead as larger documents have to be sent over the wire.

3.6. Overview

Attack	Structures and Techniques	Leakage	Complexity	Countermeasures
GLMP18 [28]	Cumulative Distribution Function Kolmogorov-Smirnov Distance Dvoretzky-Kiefer-Wolfowitz Inequality	Volume Pattern Similar Dataset	In number of possible queries	Blocked Queries impacts performance Wrap-Around Queries nullifies attack
GLMP19 [27]	PQ-Trees Binomial Distribution Order Statistics	Access Pattern Rank Information Similar Dataset	Scales in database size	Blocked Queries impacts performance Wrap-Around Queries nullifies attack
KPT20 [37]	Support Size Estimator	Access Pattern Query Pattern	System of Linear Equations scales in database size	Blocked Queries impacts performance Wrap-Around Queries impacts performance
Basic RPP-Attack	PQ-Trees	Access Pattern Known Token-Query Pairs	Scales in number of queries and database size	Blocked Queries impacts performance Wrap-Around Queries nullifies attack
Refined RPP-Attack	PQ-Trees	Access Pattern Known Token-Query Pairs	Scales in number of queries and database size	Blocked Queries impacts performance Wrap-Around Queries nullifies attack
Rank RPP-Attack	PQ-Trees	Access Pattern Known Token-Query Pairs Rank Information Similar Dataset	Scales in number of queries and database size	Blocked Queries impacts performance Wrap-Around Queries nullifies attack
Document Size RPP-Attack	PQ-Trees	Access Pattern Known Token-Query Pairs Hard Drive Info Partial Dataset	Scales in number of queries and database size	Blocked Queries impacts performance Wrap-Around Queries nullifies attack

Table 3.1: Summary of the different relevant attacks on different aspects.

Table 3.1 shows an overview of the related attacks and the ones we have designed and allows us to compare them in several aspects. As is visible, a number of the attacks use the PQ-tree structure. Furthermore, GLMP18 uses the volume pattern, while the other two use the access pattern. None of them has used known token-query pairs in their attacks. Secondly, as is visible two of the attacks use an auxiliary dataset, but none of them uses hard drive info. While GLMP19 uses rank information, the attack scales poorly with the database size to a point where it is not feasible to run. Lastly, while KPT20 is agnostic, as mentioned the system of linear equations that need to be solved scales in database size. If their assumption of duplicate removal is not existing, then this attack becomes infeasible to run. This led us to design an attack that uses known token-query pairs as an auxiliary input. We furthermore made an attack that albeit scales in the database and domain size, still can finish within acceptable time frames. Lastly, we extended these attacks by incorporating rank and document size information. The last column shows that the two tested countermeasures have a severe impact on the different attacks used. Only KPT20 can be executed with this countermeasure as long as certain assumptions are met. A more thorough analysis and comparison of the related works and the designed attacks can be found in chapter 6. We will now explain how the designed attacks are set up.

4

Novel Attacks

Now that the background and the relevant related works have been explained, the novel approach designed in this thesis can be explained in detail. First, the basic variant of the attack will be explained. Next, the refined variant will be discussed. Thirdly, we will explain the auxiliary information that we can use in our attacks and how these contribute to an even better attack. Pseudocode will be provided in the report. The attack is called *Ranged PQ-Tree Token-Query Pair Attack* or RPP-Attack for short. The source code can be found at RPP-Attack.

4.1. Basic attack

The basic attack is the attack algorithm that ensures that for each observed token, the correct query will always be in the list of options. This basic variant will first be explained from a high-level overview using the pseudo-code found in Algorithm 1. The first lines show the input and output of the algorithm. A list of at least 2 observed tokens and a list of known pairs are needed to start the attack.

The attack starts at line 8 with setting an empty list for the token predictions. At line 9, it gets the PQ-tree and the possible value estimates for each bucket. Buckets are groups of documents that are assumed to have the same value. These buckets are the leaves of the Q-node at the top. The next step is to, for each observed token, get the leakage. Based on the leakage, the algorithm gets options for an empty access pattern or based on documents in the access pattern. This happens at lines 12 to 17. Once the algorithm has found the possible list of options for a token it gets added to the list of predictions. At line 19 the mapping of documents to estimated values is retrieved. In the last line, the algorithm returns the predictions for the tokens and the document mappings. The sub-methods will now be explained in more detail.

Algorithm 1 BasicAttack

1: **Input**
2: T List of observed tokens
3: $\hat{T}$ List of known token-query pairs
4: **Output**
5: L Token to options mapping
6: M Documents to value mapping
7:
8: $predictions \leftarrow []$
9: $pqTree, correctlyOrderedBuckets, bucketToRanges \leftarrow GetPQTreePlusEstimates(tokens)$
10: **for** $token$ in T **do**
11: $leakage \leftarrow GetLeakage(token)$
12: **if** $leakage! = \emptyset$ **then**
13: $guess \leftarrow GetGuess(leakage)$
14: $options \leftarrow GetOptions(guess)$
15: **else**
16: $options \leftarrow GetOptionsEmptyPattern()$
17: **end if**
Add $token$ *and* $options$ *to* $predictions$
18: **end for**
19: $documentMappings \leftarrow GetMappingDocuments(correctlyOrderedBuckets, bucketToRanges)$
20: **return** $predictions, documentMappings$

4.1.1. Buckets and PQ-Tree

As mentioned, the first call that is made in the basic RPP-attack is to get the order from the different documents and to get the ranges of values for each bucket. This method consists of several steps. These steps are visible in algorithm 2. First, the $GetPQTreePlusEstimates$ method gets the PQ-tree from the tokens and gets a possible order from the tree. Once a possible order has been retrieved, the correct order is discovered by breaking the reflection at line 10. From this correct order ranges using the known token-query pairs can be learned at line 11. Once that is done, the method returns the PQ-tree, the order of the buckets of documents and the possible values for each of the buckets.

The PQ-tree is created using the structure described in chapter 2. In our attack, only the order of observed documents is of interest to us. This means that no records will have to be sacrificed, unlike the attack in [27], which does have to sacrifice records if not enough tokens are observed. To initialize the PQ-tree we create the set of documents that the attacker observed and put these in the tree. Next, the observed and known queries are used to reduce the PQ-tree. Once the PQ-tree has been reduced using these tokens, a Q node at the top is expected. From this Q-node we get the order by grabbing the leaves from left to right. This Q-node has either documents as leaves or P nodes which in turn have documents as leaves. There should not be multiple layers of Q nodes. If this would be the case, then some documents got grouped in Q-nodes under the top Q-node, of which we know specific order relationships. The only node that should indicate strict order is the top node. Our attack assumes that the grouping of documents under P-nodes only contains equal documents. Thus if there are more Q-nodes, there is an error. The PQ-Tree structure guarantees that this cannot happen. In our experiments, we also saw that there was only one Q-node at the top and no other places. The leaves of this Q-node were P-nodes or document identifiers.

Algorithm 2 $GetPQTreePlusEstimates$

1: **Input**
2: T List of observed tokens
3: **Output**
4: PQ Tree made from the observed tokens
5: $\hat{O}$ The correct order of the buckets of documents
6: R The possible values per bucket of documents
7:
8: $pqTree \leftarrow$GetPQTree(tokens)
9: $possibleOrder \leftarrow$GetOrder($pqTree$)
10: $correctOrder \leftarrow$BreakReflectio($pqTree, knownTokens$)
11: $ranges \leftarrow$LearnRanges($correctOrder, knownTokens$)
12: **return** $pgTree, correctOrder, ranges$

Now that a possible order of the documents is obtained, the next step is to break the reflection of the order. The current order is up to reflection because of the PQ-tree. The same problem exists in [27]. This problem exists because the PQ-Tree structure decides which of the document is relatively smaller and which one is relatively larger in the beginning and uses that during the entire operation. This breaking of the reflection is described in algorithm 3. As stated our attack has known token-query pairs as input which can be exploited to break the reflection. These known pairs are used to decide which groups of document identifiers are expected to be first in the estimated order. First, a known query with the smallest width and lowest starting point is retrieved. This known query should have at least a single document in the access pattern and is called Q_1 at line 7 in algorithm 3.

Once Q_1 has been obtained, the second query needs to be found at line 8. This second query is again non-empty and has at least a partially different access pattern than that of Q_1. Finding this query is done by finding a query which has the furthest end point and if possible a starting point which is higher than the endpoint of the first query. This second query is named Q_2. As it is also non-empty, an attacker now knows which documents are expected to be present before others.

The last step in breaking the reflection aspect is to go through the current order of buckets and see if the documents of Q_1 or Q_2 are first. If the documents of Q_1 are first, the order as it is gets returned. If the documents of Q_2 are first, the reverse order gets returned by the algorithm. This occurs in lines $9 - 14$ in algorithm 3

Algorithm 3 BreakReflection

```
1:  Input
2:      T̂   list of known token-query pairs
3:      O   initial order
4:  Output
5:      Ô   The correct order of the buckets of documents
6:
7:  Q_1 ← non-empty query with the earliest starting point and smallest width
8:  Q_2 ← non-empty query with the largest ending point
9:  for bucket in buckets do
10:     if Q_2 is present in bucket before Q_1 then
11:         return reverse order          ▷ Q_2 was before Q_1 so the order needs to be reversed
12:     end if
13:     return current order              ▷ Q_1 was before Q_2 so current order is correct
14: end for
```

To give an example of the algorithm, we look at table 4.1a. There one can see the gotten order from the PQ-Tree. The left column is the buckets of document identifiers. The right column will be filled with the minimum and maximum values for the associated bucket of documents. Using this order and the known token-query pairs, the attacker needs to get the correct order. By running the algorithm described above $Q_1 = [1, 3]$ with an access pattern of p_4, P_{10} and $Q_2 = [8, 8]$ with an access pattern of p_3, p_6 were found. If the algorithm uses these two queries, it detects that the documents from $Q2$ are present before the documents of $Q1$, so the order has to be reversed. The returned order from the method $BreakReflection$ is displayed in table 4.1b.

bucket	range
p_9, p_7	
p_1	
p_3, p_6	
p_2	
p_5, p_8	
p_{10}	
p_4	

(a) Initial order of buckets

bucket	range
p_4	
p_{10}	
p_5, p_8	
p_2	
p_3, p_6	
p_1	
p_9, p_7	

(b) Correct gotten order of buckets

Table 4.1: Order of buckets containing document identifiers before and after the order has been correctly established

4.1.2. Learning Ranges

Algorithm 4 LearnRanges

1: **Input**
2: $\hat{T}$ List of known token-query pairs
3: $\hat{O}$ The correct order of the buckets of documents
4: **Output**
5: R The possible values per bucket of documents
6:
7: $gottenRanges \leftarrow AssignInitialRanges$
8: **for** non-empty token t in $\hat{T}$ **do**
9: $accessPattern \leftarrow GetLeakage(t)$
10: $learnedRanges \leftarrow AssignRangesKnownTokensWithAccessPattern(accessPattern, t, \hat{O}, gottenRanges)$
11: **end for**
12: $removedKnownEmpty \leftarrow AssignRangesKnownTokensWithoutAccessPattern(learnedRange)$
13: $checkedRanges \leftarrow NarrowResults(removedKnownEmpty)$
14: **return** $checkedRanges$

Once the reflection is broken, the next step is to learn ranges for the gotten buckets. This is done based on the known token-query pairs. The last step is to narrow the result. The pseudo-code for this method can be found in algorithm 4.

The input for $LearnRanges$ is the list of known token-query pairs, and the correct order of buckets that were obtained earlier. At line 7, the algorithm assigns a possible range estimation for each bucket (of documents) to ensure there are options for each bucket. These first estimations already give an idea of what values these documents can have and is strongly dependent on the sparsity of the database, and the number of buckets acquired. If the database is dense and there are as many buckets as the domain is large, then the algorithm assigns exactly one value to each bucket of documents and the attack ends. The PRR-attack is then also able to assign the exact query to token matches for all tokens and have all documents perfectly mapped to the correct value. This is the same for a lot of the other attacks in the range query SSE domain.

If the database is sparse, then the PRR-attack is only able to assign a range of possible values to the buckets. This range of values is dependent on the domain size and the number of buckets that were identified. Ranges of values are assigned using an $interval$. This $interval$ is calculated by subtracting the domain size from the number of buckets present. Once the $interval$ is calculated initial values are assigned to each bucket of documents. This $interval$ allows the algorithm to get the width of possible values that a bucket can be assigned, where $width = interval + 1$. For the first bucket, the range is between [1,$1 + interval$]. For the second bucket, the range is [2, $2 + interval$] etc. An attacker knows that the first bucket cannot have a value higher than $1 + interval$ because then the rest of the buckets would not have values within the range of the domain. Every bucket also has an increase of one in its starting value as it will have a minimum of buckets before it. The width in range per bucket exists due to the sparsity mentioned before. At this stage, the algorithm is unable to exactly distinguish where the sparsity is present.

An example can be found in table 4.2 where the domain is 10 and the order gotten in table 4.1b is used. As the domain is 10 and the number of buckets is 7, the $interval$ becomes 3. As is visible, each range has a width of 4. In this example, the attacker knows that the value for the first bucket cannot be above 4 because then the last bucket would get the value of 11.

The next step is to assign the known token-query pairs that have an access pattern. The call for the tokens is done through lines 8 to 11. First, the RPP-attack gets the access pattern again. The method called at line 10 can be found in algorithm 5. This method will now be explained. The first call at line 9 in algorithm 5 is done to calculate the number of buckets that have documents of the observed access pattern. If the number of buckets is equal to the width of the token, then the buckets have single increments and single values can be assigned to the buckets. This would be the best-case scenario as then there is no doubt about any values. This occurs at lines 10 and 11.

When there are fewer buckets than the width of the token, see the last else statement at line 14 and

bucket	range
p_4	1,4
p_{10}	2,5
p_5, p_8	3,6
p_2	4,7
p_3, p_6	5,8
p_1	6,9
p_9, p_7	7,10

Table 4.2: Initial range for each bucket

15, an attacker knows that there is a gap in the values as the density is then not 100%. The PRR-attack then assigns new values to the correct buckets using a similar approach as when the attack assigned the initial values. The method $HandleTokenLeakageValueRange$ calculates the $interval$ again and assigns the new range if it is less wide than the old one. This ensures that each document still has the correct document value in the range, but the amount of possible options decreases. If there are more buckets than tokens, as visible in lines 12 and 13, the method returns empty because the token to assign for the pattern is too small.

Algorithm 5 HandleTokenLeakageValueRange

1: **Input**
2: A Access pattern of t
3: t Token
4: $\hat{O}$ The correct order of the buckets of documents
5: R The possible values per bucket of documents
6: **Output**
7: $\hat{R}$ Improved possible values per bucket of documents
8:
9: $y \leftarrow NrBuckets()$ ▷ Get the number of buckets the access pattern occurs in
10: **if** $y == len(token)$ **then**
11: $bucketToValueRange \leftarrow AssignExactValues()$
12: **else if** $y > len(token)$ **then**
13: **return** ▷ Too many buckets for token, wrong token to assign.
14: **else**
15: $bucketToValueRange \leftarrow assignNewInterval()$
16: **end if**

Once the non-empty token-query pairs have been handled, the next step is to also deal with the empty ones. This occurs at line 12 in algorithm 4. The algorithm gathers all the known tokens with empty access patterns. Then it looks at each bucket and its assigned range and removes the values in the range that are also present in the empty tokens. This is because there are no documents which can have these values. If this results in a different minimum and maximum for the bucket, then the width of the possible options for the bucket has gotten smaller and the attack assigns the new range.

Table 4.3 shows how three queries affect the initial assigned values from table 4.2. Q_1 is the first known token-query pair and can narrow down the first two buckets significantly. Instead of them having a width of four, it results in them having a width of two. This is as the interval is 1. The second query has a width of one and its access pattern corresponds to a single bucket. The attack can thus assign the token-query value to the bucket. This ensures that an attacker knows the value of this bucket with exact certainty. The last step is to handle the empty queries. Here we can filter out the value of 7 for the fourth bucket resulting in a maximum value of 6. This step is visible in table 4.3d.

To finish up the $LearnRanges$ method, it narrows the ranges of the buckets at line 13 in algorithm 4. This narrowing is done iteratively until the ranges for each bucket do not change anymore. The narrowing of the ranges is done in three phases. First, the maximum values of the ranges are investigated, and then the minimum values are checked. Once that is done, the last phase is to check if there are now ranges with a width of one.

bucket	range
p_4	1,4
p_{10}	2,5
p_5, p_8	3,6
p_2	4,7
p_3, p_6	5,8
p_1	6,9
p_9, p_7	7,10

(a) Initial range for each bucket

bucket	range
p_4	1,2
p_{10}	2,3
p_5, p_8	3,6
p_2	4,7
p_3, p_6	5,8
p_1	6,9
p_9, p_7	7,10

(b) After $Q_1 = [1, 3]$ with access pattern of (p_4, p_{10})

bucket	range
p_4	1,2
p_{10}	2,3
p_5, p_8	3,6
p_2	4,7
p_3, p_6	8
p_1	6,9
p_9, p_7	7,10

(c) After $Q_2 = [8, 8]$ with access pattern of (p_3, p_6)

bucket	range
p_4	1,2
p_{10}	2,3
p_5, p_8	3,6
p_2	4,6
p_3, p_6	8
p_1	6,9
p_9, p_7	8,10

(d) After $Q_3 = [7, 7]$ with access pattern of Ø

Table 4.3: Changes in ranges due to known queries

The limiting at the end is done as follows. The $NarrowResults$ method loops through all the ranges and gets the current end value of the bucket if it is not a single value and calls this $y1$. If it is a single value it continues. The method then grabs the range for the next bucket in the order and looks at the possible end value. This end value is called $y2$. If the value $y2$ is less than or equal to $y1$ it is known that $y1$ can be lowered. $y1$ can be lowered as the end value of a bucket has to be at least one lower than that of the next bucket in the order. $y1$ gets changed to $y2 - 1$. Afterwards, the previous buckets are checked to see if their end values can also be reduced.

The limiting from the start values is done similarly. the attack goes through all the buckets and gets the start value of the corresponding range called $x1$. Next, it grabs the start value of the adjacent bucket in the order, called $x2$. If $x2$ is less than $x1$, $x2$ is changed to $x1 + 1$. This is because every consecutive bucket needs a starting value of at least 1 higher than the starting value of the previous bucket. Once this is done for the current bucket which has $x2$, the starting value is increased and the next buckets in the order are checked.

The last step is to check if there are now ranges for buckets that have an equal starting and end value. Meaning that there is a width of 1. If this is the case, the PRR-attack sets an integer value for the bucket as the attacker now knows for certain that this bucket of documents is equal to one specific value. This increases the certainty of what the attacker knows about the database. As mentioned, these three phases happen multiple times. Once that is done, $LearnRanges$ returns the assigned ranges.

To give a concrete example of the phases described above, we would like to draw the attention of the reader to table 4.4 and specifically 4.4a. Here on the left are the document identifiers and on the right are the minimum and maximum values that these documents can have. This is the starting point for the described narrowing of the range. Table 4.4b shows that the third bucket with documents p_5, p_8 has gotten a tighter range. This happened as the maximum value for bucket 4 is equal to that of bucket three. As this is not possible the attacker knows that the third bucket has a max possible document value of $6 - 1 = 5$. As for the second bucket, the maximum is already 3 which is lower than 4, so no further changes happen here.

Table 4.4c shows the narrowing of the values at the start. Here one can see that due to bucket five having an exact value, the PRR-attack know that the starting value of the next block has to be at least 9 and for the bucket after that it has to be at least 10. These values are put into the ranges.

The last step is to do an equal check. One can see that the minimum and maximum are equal for the last two buckets so they get set to a single value.

When looking back at the starting table 4.2, it is clear that the PRR-attack has been able to assign specific values to certain buckets and with others, it has been able to significantly narrow down the range. This will in turn result in fewer options for each token.

4.1.3. Guesses

Now that the ranges for each of the buckets, the tree and the correct order have been obtained, the basic PRR-attack can proceed. The next step is to find possible options for each of the observed tokens. This occurs in lines 10 to 18 in algorithm 1. To do this we have developed two methods. One when there are documents in the access pattern, and one if there is an empty access pattern. This division

bucket	range
p_4	1,2
p_{10}	2,3
p_5, p_8	3,6
p_2	4,6
p_3, p_6	8
p_1	6,9
p_9, p_7	7,10

(a) Before *narrow ranges*

bucket	range
p_4	1,2
p_{10}	2,3
p_5, p_8	3,5
p_2	4,6
p_3, p_6	8
p_1	6,9
p_9, p_7	7,10

(b) After *limit end*

bucket	range
p_4	1,2
p_{10}	2,3
p_5, p_8	3,5
p_2	4,6
p_3, p_6	8
p_1	9,9
p_9, p_7	10,10

(c) After *limit start*

bucket	range
p_4	1,2
p_{10}	2,3
p_5, p_8	3,5
p_2	4,6
p_3, p_6	8
p_1	9
p_9, p_7	10

(d) After *equal check*

Table 4.4: Changes in ranges due further narrowing

had to be made as the gotten ranges per bucket have to be handled differently. Furthermore, a specific method for empty access patterns had to be set up to ensure that the attack does not return all options but only a correct subset. We will first explain the method to get the options for tokens which have a filled access pattern.

Get Options With Access Pattern

To get the options for tokens which have an access pattern the PRR-attack first has to get the guess by calling $GetGuess$ in algorithm 1. The pseudo-code for $GetGuess$ can be found in algorithm 6. $GetGuess$ loops over all the buckets until it finds the first bucket that contains documents from the access pattern. This happens in lines 10 and 11. From this bucket, the maximum value is retrieved to be used as the maximum starting value. This is the maximum starting value of the documents in the access pattern. The minimum starting value is obtained by looking at a previous bucket in the order if this is possible. If this is not possible the minimum starting point is equal to 1. From this previous bucket, the algorithm obtains its lowest value and increments it with 1. This increment has to happen because otherwise, the access pattern would have contained the previous bucket of documents. The algorithm also has to look at the previous bucket to ensure that it can handle gaps in the values of documents due to sparsity. Obtaining the minimum and maximum starting values after having found the first bucket occurs at lines 12 to 14.

To find the ending minimum and maximum values $GetGuess$ finds the first bucket that does not contain any documents of the access pattern. See line 15. From this bucket, it grabs the maximum end value and subtracts 1. This is the maximum possible value for the documents in the access pattern. 1 has to be subtracted because otherwise the documents of this bucket, which has no documents in the access pattern, would have been present. Next, the algorithm looks at the bucket that was earlier in the order, the last bucket which still contained documents from the access pattern, and obtains the minimum value. These are the minimum and maximum ending values for possible queries. If no bucket can be found that does not contain any documents of the access pattern the maximum end gets set to the domain size. The starting and ending values make up the *guess* that gets returned.

To give an example, we use table 4.5. If the access pattern of a token is p_{10}, p_5, p_8 then the guessed query will have a starting point $(2, 3)$ and the endpoint $(3, 5)$. While the token with access pattern of p_3, p_6, p_1, p_9, p_7 has as a guess for the starting points$(5, 8)$ and as endpoint $(10, 10)$

bucket	range
4	1,2
10	2,3
p_5, p_8	3,5
p_2	4,6
p_3, p_6	8
p_1	9
p_9, p_7	10

Table 4.5: Buckets and their associated ranges

Now that the *guess* has been created, the PRR-attack can make a list of possible options. Which is the next step in algorithm 1 at line 14. The corresponding method can be found in algorithm 7. The options are obtained by looping from the minimum start to the maximum start with an inner loop for the minimum and maximum end values. In this manner, the RPP-attack ensures that the correct query will always be present in the list of options. No matter how large or small the list might be. For the token with access pattern of p_3, p_6, p_1, p_9, p_7 the attack gets $[5, 10], [6, 10], [7, 10], [8, 10]$ as options. The actual query was $Q_x = [6, 10]$ and is indeed present in the list of options.

Algorithm 6 GetGuess

```
1:  Input
2:      A    Access pattern
3:      Ô    Correct order
4:      R    The possible values per bucket of documents
5:  Output
6:      g    A guess for A given Ô and R
7:
8:  minStart, maxStart ← 1, domain
9:  minEnd, maxEnd ← 1, domain
10: for bucket in correctlyOrderedBuckets do
11:     if start then                          ▷ First bucket that has overlapping documents
12:         minStart ← start one bucket back
13:         minStart ← minStart + 1
14:         maxStart ← end current bucket
15:     else if end then                       ▷ First bucket that has no overlapping documents
16:         maxEnd ← end current bucket
17:         maxEnd ← maxEnd − 1
18:         minEnd ← start one bucket back
19:     end if
20: end for
21: return ((minStart, maxStart), (minEnd, maxEnd))
```

Algorithm 7 GetOptionsFromPattern

```
1:  Input
2:      g    The guess for a token
3:  Output
4:      l    The list of options for g
5:  options ← []
6:  minStart, maxStart ← g.start
7:  minEnd, maxEnd ← g.end
8:  for startValue in minStart, ..., maxStart do
9:      for endValue in minEnd, ..., maxEnd do
10:         options ← options ∪ [startValue, endValue]
11:     end for
12: end for
```

Get Options Without Access Pattern

As mentioned, we also developed a method for when there is no access pattern. This method is sophisticated in the sense that it does not return all possible queries, but only queries that can be empty. The pseudo-code can be found in algorithm 8. $GetOptionsEmptyPattern$ starts by finding the end position for the first bucket. It then grabs the end value for this bucket called $y1$. We know that the database can have empty values up to the last value. Thus the method adds all possible query options until a maximum end value $y1$ to the list of options. This occurs in lines 7 and 8.

To find the next set of options, $GetOptionsEmptyPattern$ iterates over every bucket until it reaches the second to last bucket. This happens at lines 9 to 11. While looping over these buckets the method looks at each bucket. If this bucket has a tuple range, then the attack does not know which exact value the documents have. Here again, it is known that all values up to the last might not be present. This means that the attack needs to add as options all combinations of queries up to this last value within this bucket. The next step is to find all the empty query options between the current bucket and the next bucket. This is done by looking at 1 starting position further than the lowest possible value for the

current bucket. Using this start value it creates options until it reaches the maximum value of the range of the next bucket. These query options get added to the list of options. Once the code reaches the last bucket, and the last line in the algorithm, the method adds all options till it reaches the domain value.

To visualise it more we will show what queries match with an empty token for the bucket to ranges table 4.5. We will now find the options until the first bucket. This was line 3 in Algorithm 8. The max value for the first bucket is 2 thus the options list becomes $[1, 1]$. Next, we do the possible options within the buckets, which is line 5. In our example that would keep the list of options to $[1, 1]$. Now we look at the possible options between the first bucket and the second. This adds $[2, 2]$ to the list of options. When going through all the buckets until the last bucket, we end up with $[1, 1], [2, 2], [3, 3], [3, 4], [4, 4], [4, 5], [5, 5], [5, 6], [5, 7], [6, 6], [6, 7], [7, 7]$ as options. The last three buckets have consecutive integers, thus no gaps and they then also do not get added as possible options.

Algorithm 8 GetOptionsEmptyPattern

```
1: Input
2:     R   The possible values per bucket of documents
3:     S   The size of the domain
4: Output
5:     l   The list of options for a token with empty access pattern
6:
7: options ← []
8: options ← options+ options until first bucket
9: for bucket in bucketToRanges do
10:    options ← options+ options for current bucket
11:    options ← options+ options for between current bucket and the next
12: end for
13: options ← options+ options from the last bucket
```

After having gotten the options for all the tokens the attack needs to get the values for the documents in the buckets. This is done at line 19 in algorithm 1. If the range is a tuple, the RPP-attack sums the starting and ending values and divides them by two. This resulting value is then assigned to the documents in the buckets. This mapping, together with the predictions for the tokens gets returned from the attack.

4.2. Refined Attack

Now that the basic RPP-attack is explained, the refined RPP-attack can be explained. For the refined variant, we use a similar principle as Damie et al. did [18]. We both use a *refine speed* and we both increase our knowledge based on the matches that are made. Furthermore, both add a score functionality to rank the matches [18]. Just the knowledge an attacker improves upon is different. The pseudo-code for the attack is visible in algorithm 9. Here the red text is the code added for the refined variant.

The attack starts in the same way as the basic attack. The refined RPP-attack gets the buckets and the guessed ranges and gets the guess for each of the observed unknown tokens. After this, the method deviates. For each prediction, a score gets added. This score is based on how far the minimum and maximum starting and end points are apart. The closer they are, the more certain the attack is of the prediction and thus the lower the score. For the observed tokens with empty access patterns, the attack returns the maximum value for the score. This score is used to order the predictions at line 23.

If the number of unknown tokens is less than the previously defined refine speed, *s*, then the refined RPP-attack assigns the predictions of the leftovers to the list of predictions. The method then returns the predictions and the estimated values for each document similar to the basic attack. This occurs at lines 25 and 26. If this is not the case, the attacker wants to further refine its knowledge. To refine the different ranges for the buckets the algorithm finds the top s tokens which have the lowest score. The refined variant then assigns a query to the token. This is done at line 31 by choosing the middle values of the minimum and the maximum of the guesses. Next, the mapping of this ”known” token-query pair gets handled. This ”known” token-query gets assigned as described earlier using the method called at lines 32 and 33. This ensures that the ranges per bucket become smaller and that there is a higher

chance of having an exact correct match. Yet this method does mean that the refined variant loses the guarantee of always having the correct query present in the list for the tokens. As mentioned earlier, an assigned query for a token can result in more buckets than the width of the query. If this is the case the attack takes the middle value again but ceils this value. Meaning that the width increases. If this one is also not wide enough, the attack notifies the user and ends the attack. After the successful assignment, the attack adds the prediction for this token to the final list of predictions. Once that is done it keeps doing this for the remainder of the tokens which do not have a mapping yet.

Algorithm 9 RefinedAttack

Input
  T List of observed tokens
  $\hat{T}$ List of known token-query pairs
  s Refine speed for the refined algorithm
Output
  L Token to options mapping
  M Documents to value mapping

$predictions \leftarrow []$
$unknownT \leftarrow T$
$pqTree, correctlyOrderedBuckets, bucketToRanges \leftarrow GetPQTreePlusEstimates(tokens)$
while $unknownT! = \emptyset$ **do**
  $tempPredictions \leftarrow []$
  for $token$ in T **do**
    $leakage \leftarrow GetLeakage(token)$ $GetGuess(leakage)$
    if $leakage! = \emptyset$ **then**
      $guess \leftarrow$
      $options \leftarrow GetOptions(guess)$
      $score \leftarrow GetScore(guess)$
    else
      $options \leftarrow GetOptionsEmptyPattern()$
      $score \leftarrow maximumValue$
    end if
    Add $token$ as key and $options$ and $score$ as values to $tempPredictions$
  end for
  sort $tempPredictions$ ascending order based on $score$
  if $|unknownT| \leq s$ **then**
    $predictions \leftarrow predictions \cup tempPredictions$
  else
    for top s token mappings t in $tempPredictions$ **do**
      $accessPattern \leftarrow GetLeakage(t)$
      $guess \leftarrow GetGuess(accessPattern)$
      $guessedQuery \leftarrow GetGuessedQuery(guess)$
      $bucketToRanges \leftarrow$
$AssignRangesKnownTokensWithAccessPattern(accessPattern, t, \hat{O}, bucketToRanges)$
      $predictions \leftarrow predictions \cup t$
      $unknownT \leftarrow unknownT \setminus t$
    end for
  end if
end while
$documentMappings \leftarrow GetMappingDocuments(correctlyOrderedBuckets, bucketToRanges)$
return $predictions, documentMappings$

4.3. Auxiliary Information

Now that the two main attacks have been defined, the three attacks using auxiliary datasets can be explained. This auxiliary information can give an attacker more knowledge and increases the accuracy of the attacks. The first is based on the volume of the documents, the second is based on the ranks of the data and the auxiliary dataset, and the third is the rank and volume attacks combined. For both attacks, the auxiliary dataset is used to refine the ranges for the different buckets at the initial setup step. The rest of the algorithm is the same as in the basic and refined attacks. This allows for an easy extension in case researchers discover an even better way to handle auxiliary information.

4.3.1. Volume Leakage

Documents, both encrypted and unencrypted, have a volume on the hard drive. An attacker can try to link server and plaintext documents based on the volume. We assume in our experiments and setup that the volume for identical documents on both the server documents and in the known documents set are equal in size. The attack should our idea also work if this is not the case, but then the accuracy will go down due to a higher chance of a wrong match. Then the links need to be adapted so that they can work on the closest matches. Furthermore, due to the need of linking documents based on volume, only a partial document set can be used.

To match the different volumes of the known document set to the server set, the attack looks at different combinations. If the volume auxiliary RPP-attack can find a unique volume on the server, then there could be a unique match in the known dataset. If the found volume is not present in the known document set, then the attack cannot link it. It is possible that volumes in themselves are not unique, but that a sum of them is. Thus the attack sums all combinations up to three documents per bucket. If a summation is unique on the server and that is also present in the known set then the attack can still link the actual keyword value to the bucket. We chose a maximum length of 3 for combinations due to run time. Increasing it higher will result in higher accuracy, but we deemed this increase to be less important than the increase in run time. Being able to exactly match values to buckets increased the accuracy of the attack significantly. The experimental results of this can be viewed in chapter 5.

4.3.2. Rank Leakage

The last extension of the attack using auxiliary information is the extension with rank leakage. Here the attacker will use the ranks of the different values in the known dataset to try and match them to the ranks on the server. To do this the rank PRR-attack finds both the ranks of the known dataset for each value in the domain and the ranks based on the documents in the buckets on the server side. To then improve the results, it looks at each bucket and grabs its rank and the previous rank called ub and lb. Next, the attack finds the lower bound on the auxiliary set that is as close to lb but at least lower called $estimated\_lb$. It does the same for ub by finding the closest rank in the ranks of the auxiliary set as long as it is larger than ub which is defined as $estimated\_ub$.

From these new estimations, the rank RPP-attack can retrieve the actual integer values from the auxiliary ranks. If these new values are within the range of the already assigned range and improve the width, then the new values get assigned. Using this extension, for both the basic and the improved variant, the two variants will not be able to guarantee that the correct query is present in the list for the token unless the full partial dataset is known. The advantage is that it can handle a similar dataset as well as a partial dataset. The results section shows the effectiveness of the extension.

5

Experiments and evaluations

This section will go over the results. These will include several sections. Firstly, datasets and pre-processing steps. Followed by results of the RPP-attack and other attacks, including run-time. Lastly, countermeasures and their impacts on the performance will be stated.

5.1. Datasets

The first dataset, the density dataset, was chosen to see how well the attacks performed on sparse and dense databases. It is an artificial dataset created during the experiments. A thorough discussion and explanation of what these results mean and their impacts is discussed in section 6

The Enron and Apache Lucene datasets were chosen to see how the attacks perform on finding emails according to their dates, which could be a beneficial usage of SSE. Lastly, a lot of range query attacks check their performance on the medical database HCUP [29]. We do not have access to this dataset. We found a dataset with similar characteristics, the banking UCI Machine Learning dataset [51]. A summary of the last three datasets can be found in table 5.1. No other dataset such as Wikipedia was chosen because these datasets resulted in distinct result and the general Wikipedia pages are not very suitable to extract numerical values from.

Dataset	Used Nr Documents	Density	Domain Size
Enron	2299	0.88	99
Lucene	730	0.85	100
UCI	27127	0.98	78

Table 5.1: Summary of dataset

5.1.1. Density

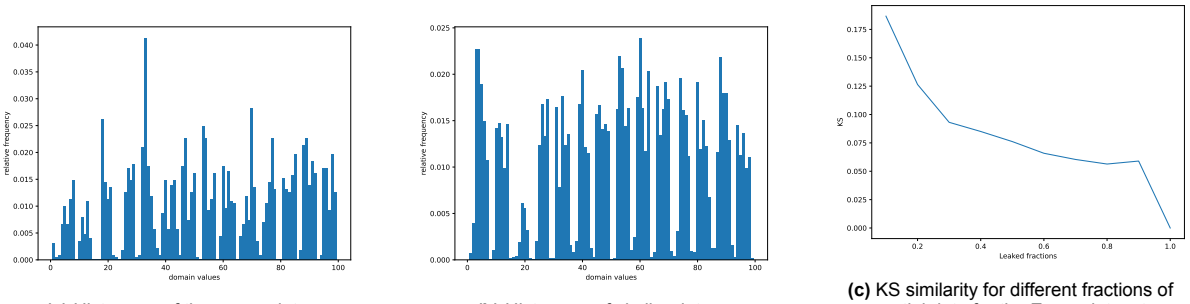
It is important to evaluate attacks on different densities as previous attacks have shown that the density of a database can impact the results greatly [37]. To do this with more control, we created our own database. This database has a domain of 100. For every density we check, a random subset of values is chosen from the domain $[1, 100]$. For every value that was chosen, four documents are generated with a random document volume between 1 and 100, with an additional $\frac{2}{3}$ chance that an extra document gets added. Lastly, in the generation of the database, we add five documents with the highest value. This choice of values and generation of documents was done to create a relatively uniform database. We ran the experiments, including document generation, 25 times for each of the attacks and took the average of the performance metrics to get the results.

5.1.2. Enron

The second dataset used is the Enron dataset [17]. This is a database of emails from about 150 employees at the Enron company [17]. It consists of 30199 email files. To make these usable for our experiments we looked at the dates of when these e-mails were sent. For every email, the date was retrieved and then normalized in such a way that the earliest date in the set gets the value 1. The

rest of the dates will be relative to the first date. The next step is to obtain the documents which were sent in the first 100 days to get a maximum domain of 100. Using the Enron dataset, this resulted in a domain of 99. The density of the document set is 0.88. From the gathered documents, which have the dates as their keywords and, as volume, the actual volume of the files in bytes, a dataset was created. For the 100 chosen days we get 2299 documents. Dates were chosen as keywords because a user might want to find emails based on them. If a similar dataset was needed, we used the first 99 days of the next year. This similar document set had 7527 documents and a density of 0.9696.

Figurese 5.1a and 5.1b show the histograms of both the used server and similar document sets. The Kolmogorov–Smirnov (KS) similarity of these two datasets is 0.1010. Although the KS score is low, there are visible differences in gaps and peaks. Fig. 5.1c shows average KS similarity values for different fractions of the partial dataset. It starts at a KS of around 0.2 and goes down quickly until half of the documents are leaked, after which it reaches a plateau The last jump to 0 can be explained because then there is no difference in the datasets.



(a) Histogram of the server data

(b) Histogram of similar data

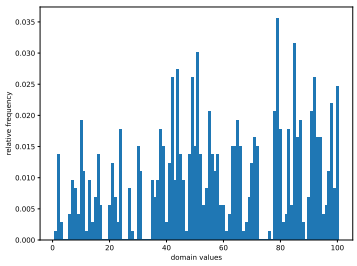
(c) KS similarity for different fractions of partial data for the Enron dataset

Figure 5.1

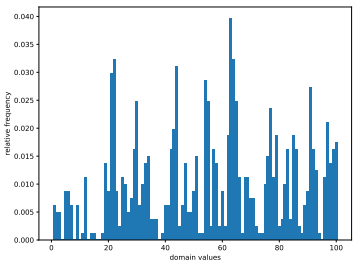
5.1.3. Lucene

The third dataset is that of the Apache Lucene email database [1]. We retrieved the emails sent by ”java-user” starting in 2002 till the end of 2004. For this database, dates from the emails were obtained and the first date was set to 1, while the rest of the dates are relative to this point. The selected emails had a value between 1 and 100. Furthermore, the volume of the documents was set to the byte size of the email contents. This resulted in a database with a domain of 100, a density of 0.85 and 730 documents. This is fewer documents than Enron and allowed us to see how the attack performs on a dataset that is less dense and has fewer documents. The similar document set was obtained by using the documents which were sent in the first 100 days of the next year. This document set had 806 documents and a density of 0.92

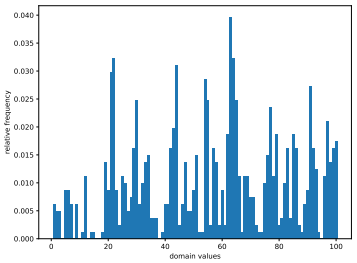
The histograms for both the server and similar datasets for the Lucene dataset can be viewed in Fig. 5.2. The KS similarity for the two datasets is 0.1. Just like with the Enron dataset, the gaps in document values are at different locations. Looking at the similarity of the partial dataset, one can see that it starts higher than the Enron dataset, at 0.4. Then it goes down quickly to a KS of 0.1 after which it plateaus. This occurs when around half the documents are leaked. In the end, it drops down to 0 because the datasets are identical.



(a) Histogram of the server data



(b) Histogram of similar data



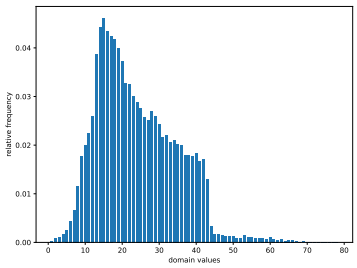
(c) KS similarity for different fractions of partial data for the Lucene dataset

Figure 5.2

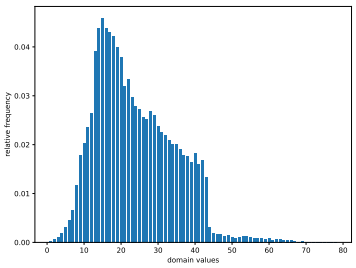
5.1.4. UCI Machine Learning Data

The last used dataset was the bank marketing dataset [51] from the UCI Machine Learning Data repository [52]. From this dataset, we looked at the ages of the different users. The minimum is 18 and the maximum is 95. These values were normalized. This leaves us with a domain of 78 and a density of 0.98 and a total of 45212 files. 60% of these documents were put on the server and the remaining 40% was used for the similar dataset. This domain is lower, but the large number of documents allows us to see how the attacks perform on such a dataset. Furthermore, it allows us to look at the run time of the algorithms and see where the bottlenecks lie. For this dataset, we normalized the values such that the lowest was 1.

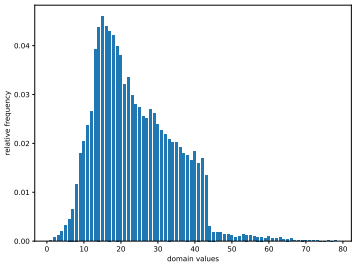
Looking at the two histograms for the UCI dataset in Fig. 5.3 one can see that they are practically identical. Furthermore, the KS similarity for these two sets is 0.05. When looking at the partial dataset figure, one can see that the similarity starts at 0.12 showing that having very little known data results in having a similar dataset. From there it goes down sharply until it reaches around 0.06 - when half of the documents are leaked. In the end, it drops to 0 for the same reasons previously mentioned.



(a) Histogram of the server data



(b) Histogram of similar data



(c) KS similarity for different fractions of partial data for the UCI dataset

Figure 5.3

5.2. Metrics

Throughout the experiments, we use six metrics. The first one is the *Fraction of correct query present in list* also called the *raw accuracy*. This is a metric that shows which fraction of the tokens had the correct query in the list of options. Here the higher the better. The second metric is the percentage of correct one-to-one matches for the query and tokens. Since we want as many one-to-one matches as possible, the higher the value, the better. The third and fourth metrics are regarding the average number of options and the average options when the correct query is present in the list. These two allow us to see if there are significantly more options to ensure a raw accuracy of 1. As we want an attacker to be as certain as they can be, the fewer options there are, the fewer the attacker has to choose from and the better it is. Lastly, we have the mean absolute error (MAE) and the mean squared error (MSE), for both a score of 0 is the best. These show how well the document values can be recovered. We use the MSE to see if we have many large errors or only small ones. The MAE allows us to see more accurately how far off the attacks are. For each metric, the standard deviation (SD) is also displayed by vertical bars.

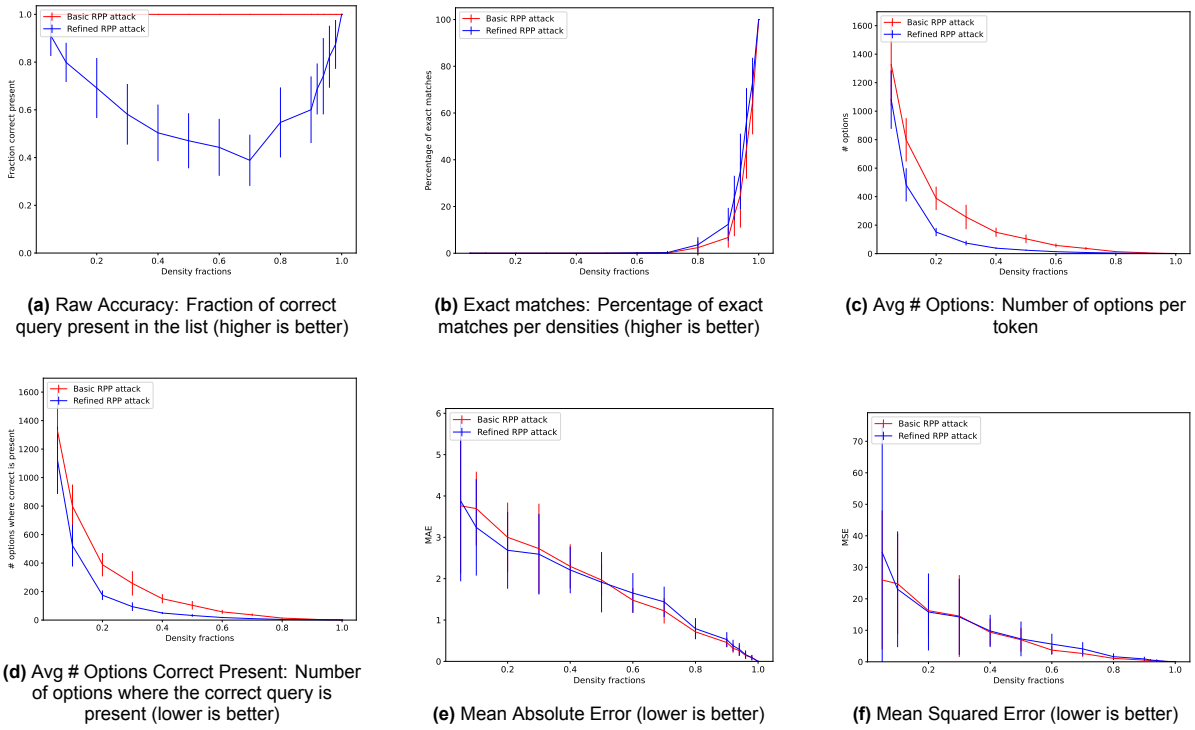
5.3. Results of Attacks on Different Datasets

We ran each experiment of the first three datasets, 25 times to get an average. Furthermore, in the case where the top node was not a Q node we did it again. This happened very irregularly and is due to the observed queries not having any overlap. In the case of an attacker, they could wait until more queries were observed to guarantee that the top node was a Q node and that a solution could be found. For the last dataset, the banking dataset, each experiment was run 5 times due to the increased run-time.

Several parameters are set throughout the experiments unless stated otherwise. The number of known queries was set to 30 for the direct comparison between KPT20, GLMP18 and GLMP19, auxiliary attacks and density experiments. The number of known queries was set to 60 for the query distribution experiments. This parameter changes when we look at how the number of known token-query pairs impact the performance. The fraction of observed queries is set at 0.15. For the GLMP18 we chose to have all the server data available, while for the GLMP19 attack we chose to only have 80 percent of the server documents available. We have also run experiments to see how different leaked fractions impact GLMP18 and GLMP19, there the known data differs per experiment. KPT20 has no specific settings.

5.3.1. Density

Comparison of RPP-Attack



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)

(b) Exact matches: Percentage of exact matches per densities (higher is better)

(c) Avg # Options: Number of options per token

(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

(e) Mean Absolute Error (lower is better)

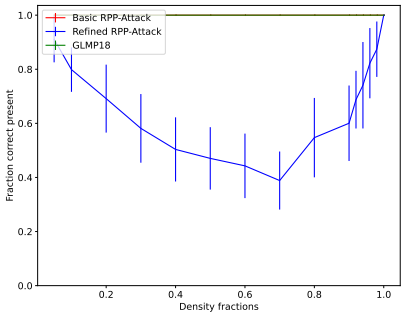
(f) Mean Squared Error (lower is better)

Figure 5.4: Comparisons of metrics for different density values for own attack

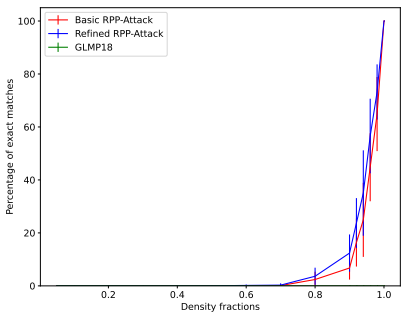
In Fig. 5.4 one can see the metrics for different densities for the refined and basic RPP-attacks. Fig. 5.4a shows that for the basic attack the correct query is always present while for the refined attack, it is present for a large fraction, then goes down to around 0.6 after which it goes up to 1 when the density is 1. Fig. 5.4b, shows that the blue line is higher throughout the experiment showing that the refined variant can have more 1-1 matches than the basic variant. In figures 5.4c and 5.4d, it is visible that the refined variant has fewer options than the basic variant. The difference goes down the higher the density is. Lastly, in Fig. 5.4e 5.4f, it is visible that the MEA and the MSE go down in almost a linear line with little difference between the two variants of the attack. For all metrics other than for the raw accuracy and the exact matches, the SD goes down as the density increases. For the raw accuracy, the SD remains stable. Regarding the exact matches, the SD increases when exact matches occur

after which it stabilizes and becomes 0 for a dense database.

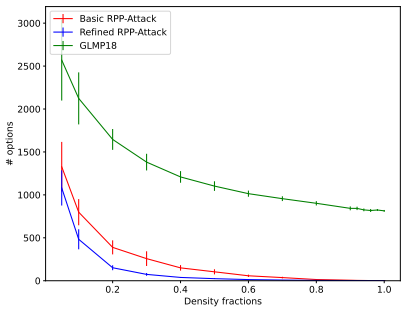
GLMP18 vs RPP-Attack



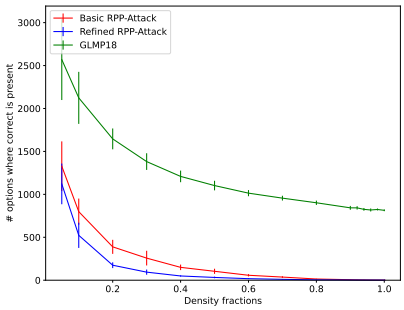
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



(b) Exact matches: Percentage of exact matches per densities (higher is better)



(c) Avg # Options: Number of options per token

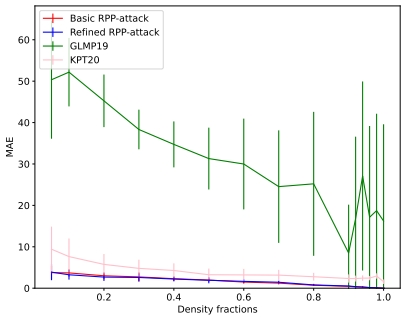


(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

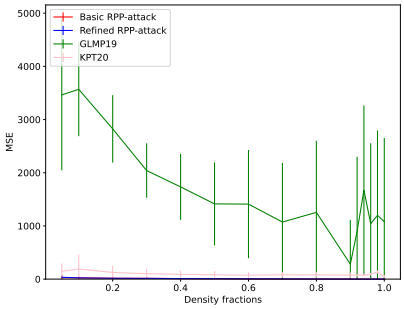
Figure 5.5: Comparisons of metrics for different density values for own attack compared to GLMP18

Fig. 5.5 visualises how well the GLMP18 attack performed on different densities. Fig. 5.5a shows that the GLMP18, just like the basic variant, can have the correct query for each density and has an SD of 0, yet it has no exact matches, as visible in Fig. 5.5b. The number of options is higher for GLMP18. It starts with having 2500 options after which it goes down to around 1000 options for each token. The maximum difference between our attacks and GLMP18 is more than 1000 options. Just as with the RPP-attacks, the SD regarding the number of options goes down as the density increases.

GLMP19 vs KPT20 vs RPP-Attack



(a) Mean Absolute Error (lower is better)



(b) Mean Squared Error (lower is better)

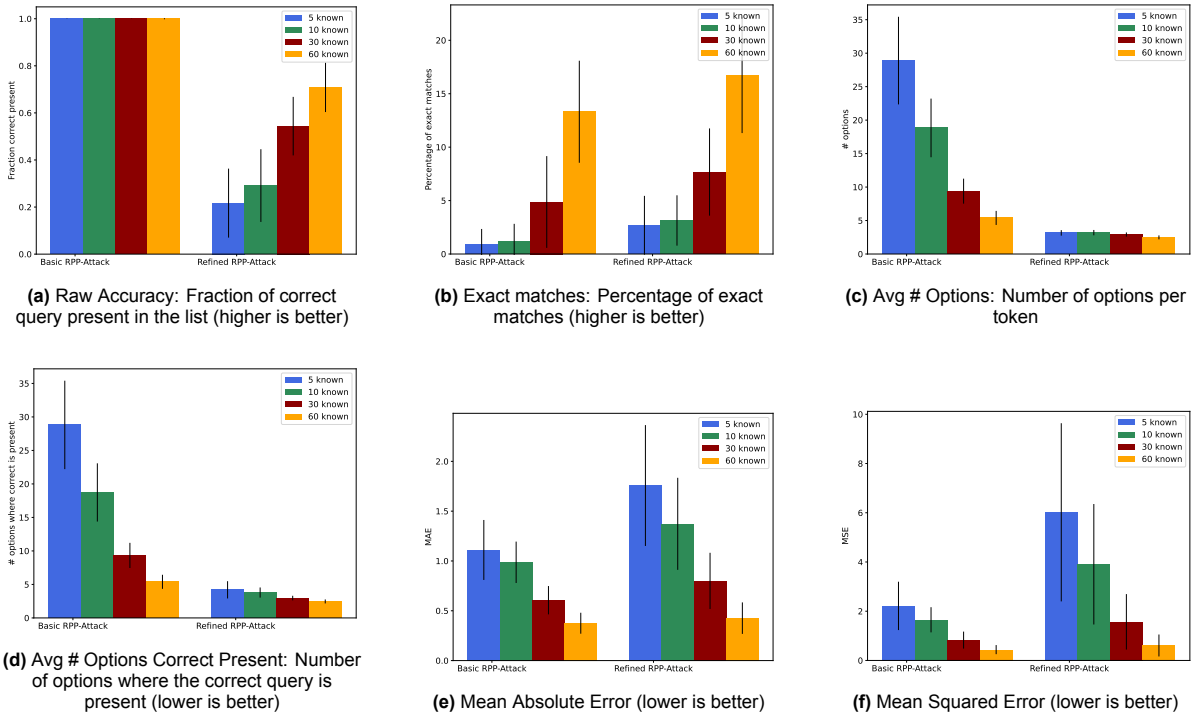
Figure 5.6: Comparisons of metrics for different density values for own attack compared to GLMP19 and KPT20

In figures 5.6a and 5.6b, it is visible that the GLMP19 attack has the highest MAE and MSE. For GLMP19, the average MAE does not get below 10, yet looking at the SD, it is visible that the values

can get relatively low and high. The average MAE of the other three does not go above 10. For all attacks, there is a downward trend for the MAE and MSE with a higher density. The refined and basic attacks have the lowest scores. For the KPT20 and RPP-attacks, the SD also decreases with a higher density.

5.3.2. Enron

Comparison of RPP-Attack



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)

(b) Exact matches: Percentage of exact matches (higher is better)

(c) Avg # Options: Number of options per token

(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

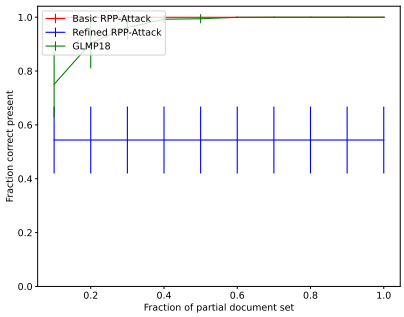
(e) Mean Absolute Error (lower is better)

(f) Mean Squared Error (lower is better)

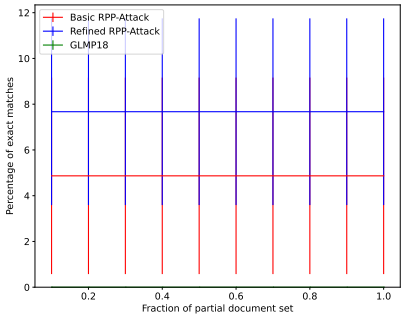
Figure 5.7: Comparisons of metrics for the different number of known queries for our basic and refined attacks.

In Fig. 5.7 the results from our basic and refined attacks using multiple different known queries on the Enron dataset are visible. Fig. 5.7a, shows that for any different number of known token-query pairs the basic attack will always have the correct query present, shown by the SD also being equal to 0, while with the refined variant this is not the case. It scores better with an increasing number of pairs. Looking at the percentage of exact matches we can see that knowing more queries results in having more exact matches for both the basic and refined variants. Here the refined variant scores better than the basic variant. The SD for both variants does not go down with more pairs, instead, it increases. Looking at the number of options per token in Figures 5.7c and 5.7d, more known queries result in fewer options. For the basic variant, this goes down from almost 30 to 7. For the refined variant, the average number of options is slightly higher if the correct query is present compared to overall. As the number of pairs increases, the SD for the number of options goes down. Lastly, in the last two figures, one can see that both the average MAE and MSE, as well as the SD, go down with more known pairs. For this metric, the refined attack scores worse than the basic variant.

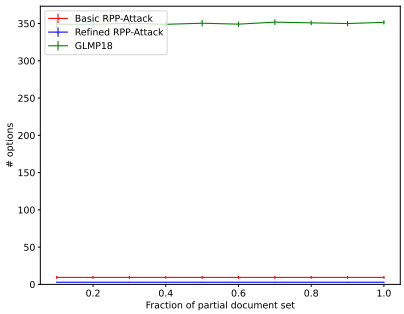
GLMP18 vs RPP-Attack



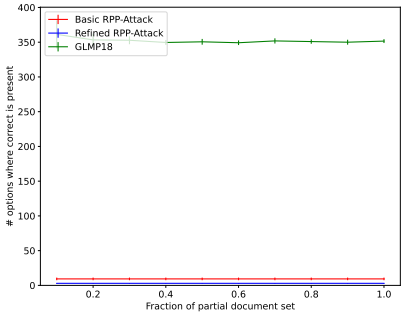
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



(b) Exact matches: Percentage of exact matches (higher is better)



(c) Average number of options for each token

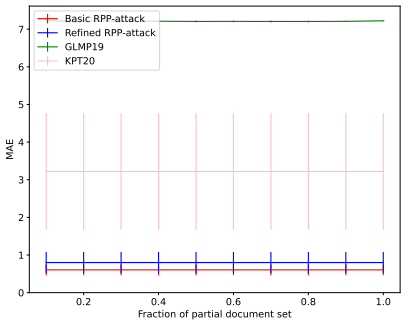


(d) Average number of options for each token

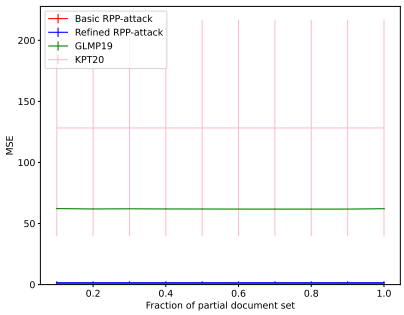
Figure 5.8: Enron GLMP18 vs own attacks comparisons for different fractions of partial data

In Fig. 5.8 the RPP-attacks are compared to the GLMP18 attack. Fig. 5.8a shows that half of the dataset needs to be leaked for the GLMP18 attack to be able to have the correct query in the list for the observed token. The SD also goes down as more of the document set is available. The GLMP18 attack also has no exact matches while our attack does, as visible in Fig. 5.8b. It is visible that GLMP18 has a higher average number of options in figures 5.8c and 5.8d. If the correct query is present, there are about 350 options. The SD is the largest for GLMP18 and remains relatively stable for the different leaked fractions of data.

GLMP19 vs KPT20 vs RPP-Attack



(a) MAE



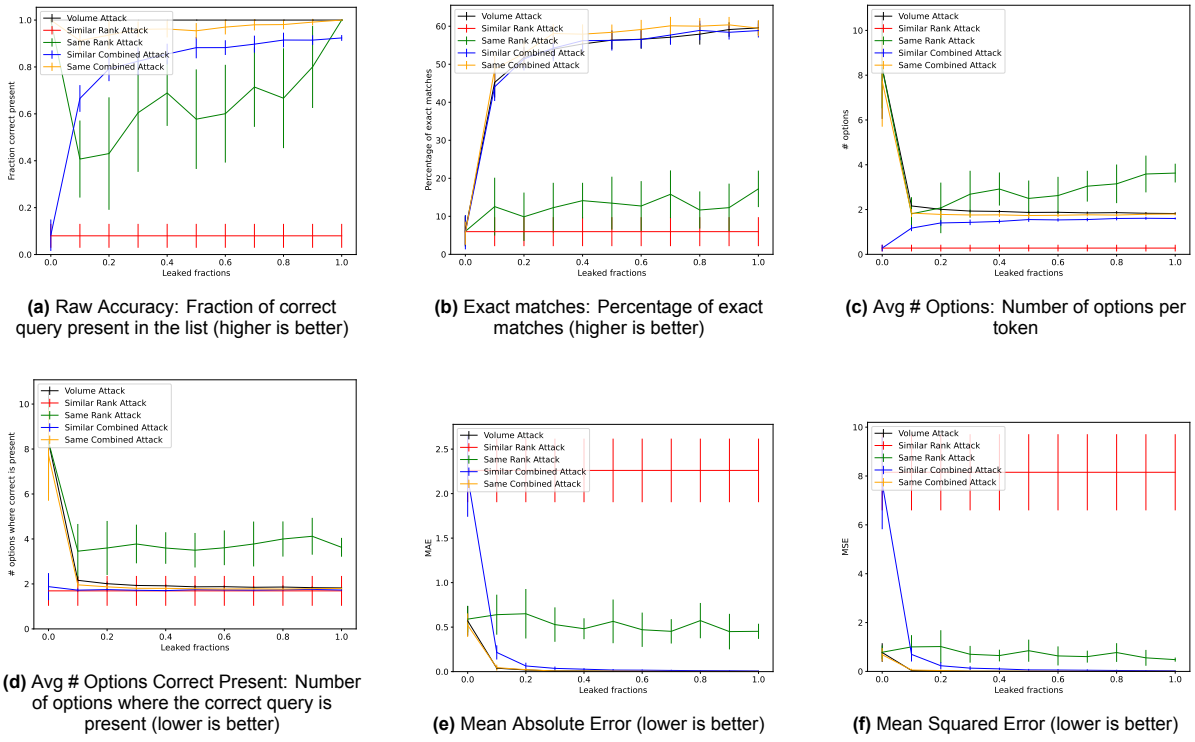
(b) MSE

Figure 5.9: Enron comparison of own attacks vs GLMP19 vs KPT20 for different fractions of partial data

For the comparison where GLMP19 has an increasing fraction of documents available, the results for KPT20 and the RPP-attacks will have a static line and SD as they are independent of the documents known. Such a setup was done to see if and when the GLMP19 attack would outperform the others. This graph setup is identical to the Lucene and UCI datasets.

Looking at the differences between GLMP19, KPT20 and our attacks in Fig 5.9a and 5.9b, we can see that the MAE of GLMP19 is stable at around 7, no matter what fraction of the document set is leaked. The MSE is also stable at 60. Furthermore, the SD is the lowest for all attacks. The KPT20 scores a low MAE but a high MSE and the largest standard deviation. The MAE of KPT20 is a bit higher than 3 and is constant as it requires no document leakage. Our attacks have the lowest MAE and MSE.

Auxiliary Data



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)

(b) Exact matches: Percentage of exact matches (higher is better)

(c) Avg # Options: Number of options per token

(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

(e) Mean Absolute Error (lower is better)

(f) Mean Squared Error (lower is better)

Figure 5.10: Comparisons of metrics for the different auxiliary attacks with the basic variant

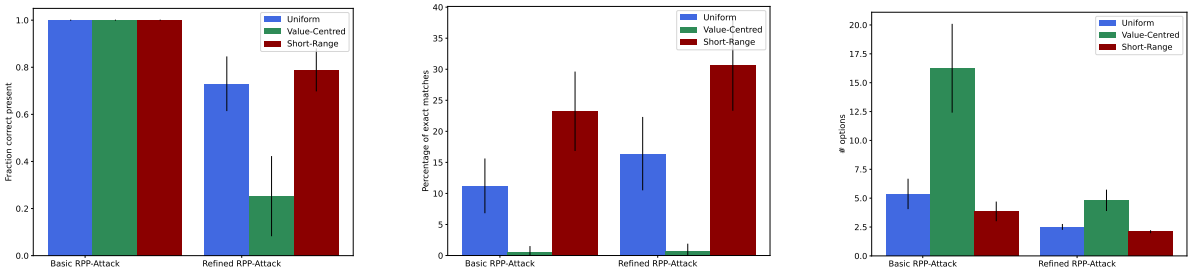
The results for the auxiliary extension for the basic attack can be seen in Fig. 5.10. Fig. 5.10a shows that the volume extension attack guarantees that the correct query is present. It also results in an increase of exact matches to above 50 percent with 20 percent of known documents with a small SD as visible in 5.10b. Furthermore, it results in a decrease in the number of options, MAE and MSE as the number of known documents increases as can be seen in see figures 5.10c, 5.10d 5.10f and 5.10f. The SD for this variant is smaller compared to the other attacks

For the two rank auxiliary attacks, we can see that the similar dataset variant has a low raw accuracy. If the rank attack uses the partial dataset, it is unable to guarantee that the correct query is always present. For the partial data rank attack, one can see that the percentage of exact matches goes up and that the MAE and MSE go down with a higher partial document set. The SD for these two attacks remains relatively stable for each fraction.

The partial dataset combined attack outperforms the other attacks on almost all metrics. It has a low MAE and MSE and a low number of options. Furthermore, it also results in an increase in exact matches compared to only having volume as the auxiliary dataset. Lastly, the similar combined attack follows the other attack curvature-wise but remains below it in all metrics. For the similar combined variant, we can see that the SD is similar to that of the volume bases attack instead of the rank attack.

The refined auxiliary attacks follow a similar result distribution as the basic auxiliary attacks as can be seen in Fig 5.10 in appendix B. The only difference is that the MAE and the MSE score worse in the refined variants.

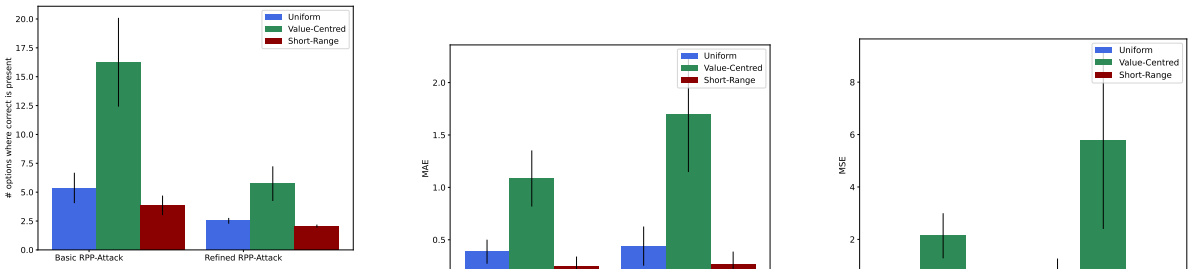
Query Distribution



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)

(b) Exact matches: Percentage of exact matches (higher is better)

(c) Avg # Options: Number of options per token



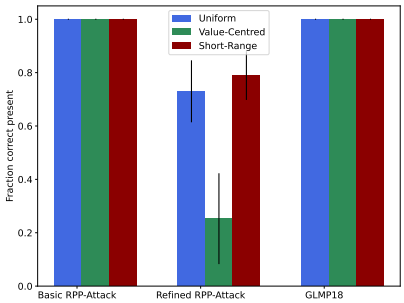
(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

(e) Mean Absolute Error (lower is better)

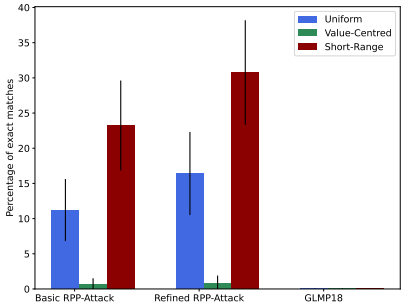
(f) Mean Squared Error (lower is better)

Figure 5.11: Comparisons of metrics for different query distributions for our basic and refined attacks.

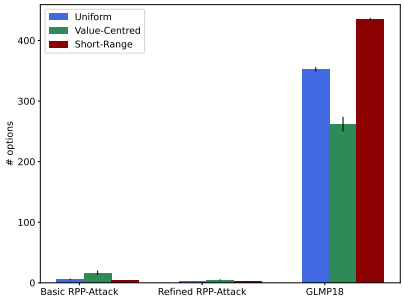
In Fig. 5.11 the results for how the attack performed on different query distributions are visible. For all three distributions, the basic attack has the correct query for each token. For the refined variant, the value-centred distribution scores lower than the other two and has the largest SD, with a difference of 0.4. Regarding the number of exact matches, one can see that the short-range is the best, even taking the SD into account, followed by the uniform and lastly the value-centred distribution. Looking at the number of options in figures 5.11c and 5.11d, it is visible that for both variants the value-centred distribution has more than double the average number of options and the largest uncertainty following from the SD, compared to the other two distributions. For all three distributions, the refined variants have fewer options. The MAE and MSE are the highest for the value-centred distribution as visible in figures 5.11e and 5.11. The uniform distribution is again in the middle and the short-range distribution has the lowest values of MAE and MSE. Here the uncertainty following from the SD has an identical distribution.



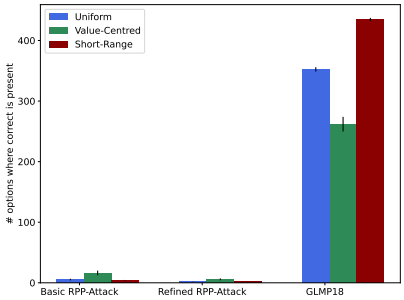
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



(b) Exact matches: Percentage of exact matches (higher is better)



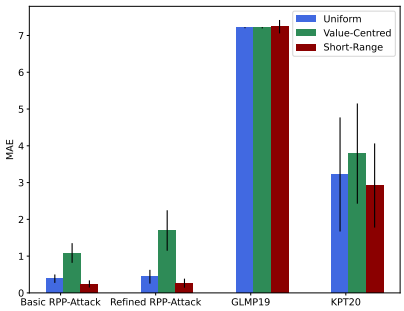
(c) Average number of options for each token



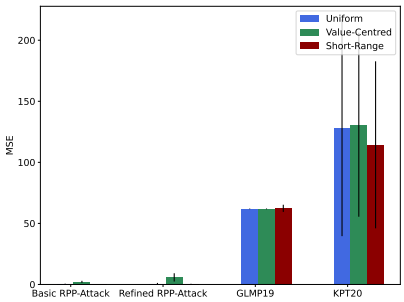
(d) Average number of options for each token

Figure 5.12: Enron GLMP18 vs own attacks comparisons for different query distributions

For all three distributions, the GLMP18 attack has the correct query for each token as visible in 5.12a. Fig. 5.12b shows that GLMP18 has no 1-to-1 mappings. Looking at figures 5.12c and 5.12d, one can see that GLMP18 has an inverse order to our attacks. For the GLMP18 attack, the value-centred distribution has the fewest possible options, the uniform distribution is in the middle, and the short-range distribution closes the group. The value-centred distribution does have the highest SD for GLMP18. The average number of options is more than 10 times as high for the GLMP18 attack compared to the PRR-attacks for all distributions.



(a) MAE (lower is better)



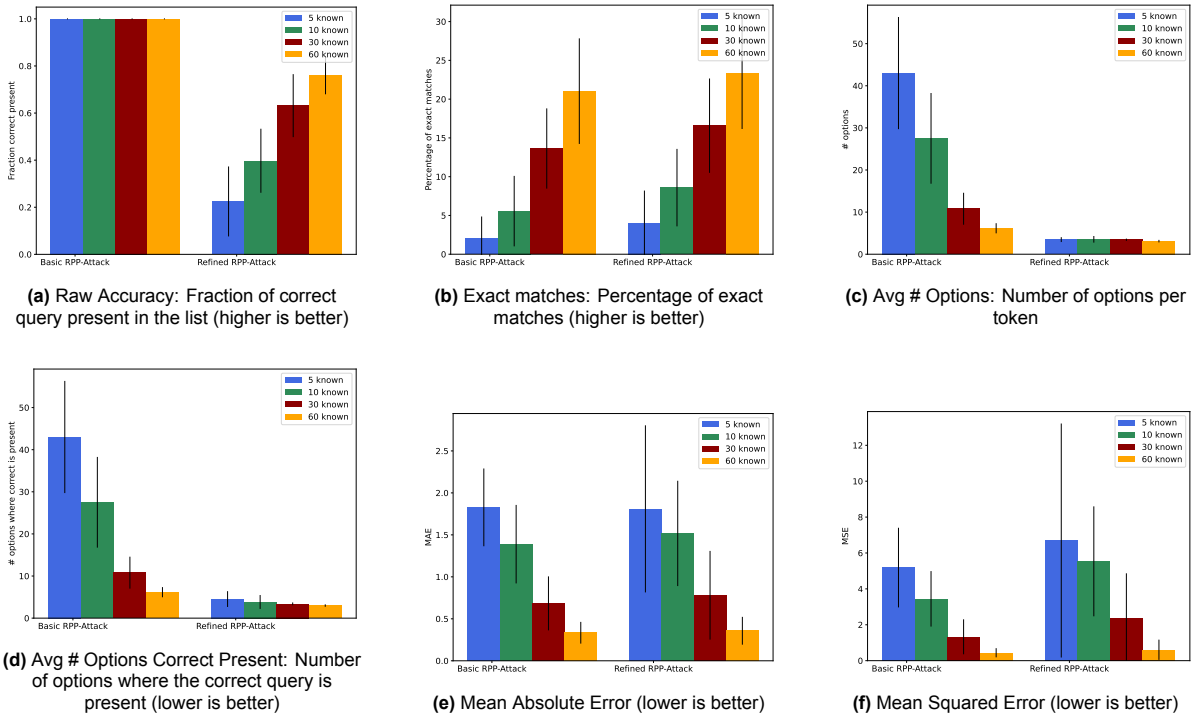
(b) MSE (lower is better)

Figure 5.13: Enron comparison of own attacks vs GLMP19 vs KPT20 for different query distributions

Looking at the last set of results for the Enron dataset, visible in Fig. 5.13a and 5.13b, one can see that GLMP19 has the worst performance for all three distributions which score effectively the same. Yet it has the smallest SD out of the attacks. The KPT20 attack follows the same order in performance of distributions as our attack, yet the SD is the largest of all the attacks.

5.3.3. Lucene

Comparison of RPP-Attack



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)

(b) Exact matches: Percentage of exact matches (higher is better)

(c) Avg # Options: Number of options per token

(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

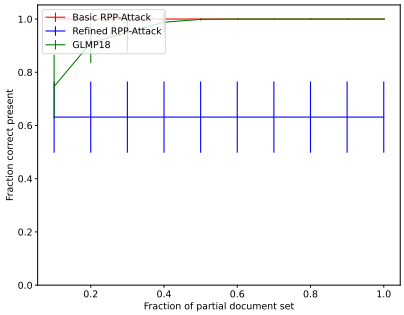
(e) Mean Absolute Error (lower is better)

(f) Mean Squared Error (lower is better)

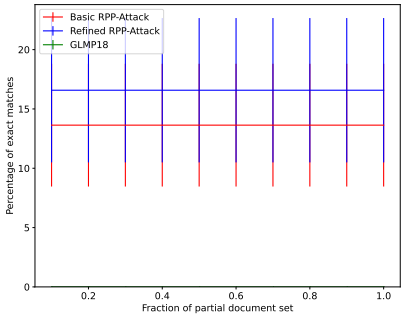
Figure 5.14: Comparisons of metrics for the different number of known queries for our basic and refined attacks.

Fig. 5.14a shows that the more pairs are available, the higher the raw accuracy for the refined variant. For both variants, the average percentage of exact matches goes up to more than 20 percent for the largest number of known pairs, as visible in Fig. 5.14b. The largest number of known pairs also results in the largest SD. Looking at figures 5.14c, and 5.14d, it is visible that the more an attacker knows, the fewer average options are possible for the tokens and the lower the SD. The average number of options goes down from more than 40 options to less than 10. For the basic variant, there is a large difference, yet for the refined variant, this difference is smaller and only visible in Fig. 5.14d. Lastly, figures 5.14e, and 5.14f, show that for both the basic and refined variants the average MAE and MSE are below 2 and 7 respectively, and while the difference in averages does not differ much, there is a difference in SD, which is larger for the refined RPP-attack. The MAE and MSE go down as more pairs are known.

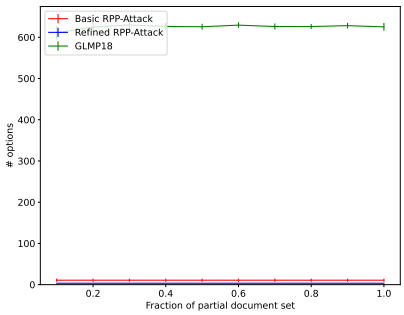
GLMP18 vs RPP-Attack



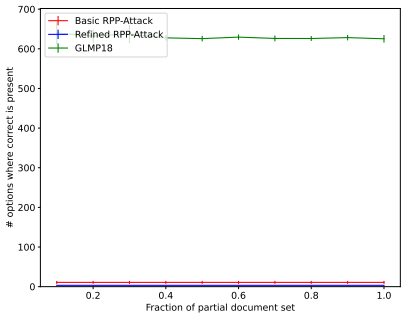
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



(b) Exact matches: Percentage of exact matches (higher is better)



(c) Average number of options for each token

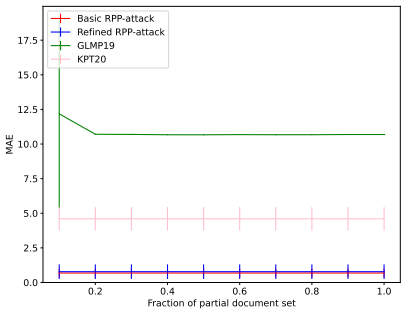


(d) Average number of options for each token

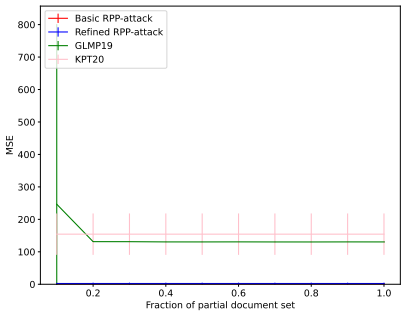
Figure 5.15: Lucene GLMP18 vs own attacks comparisons for different fractions of partial data

Fig. 5.15 shows the results for GLMP18 compared to the RPP-attack. Fig. 5.15a indicates that half of the document set needs to be known before GLMP18 has a raw accuracy of 1 and before the SD also becomes 0. If more than 10 percent of the dataset is known, GLMP18 outperforms the refined variant. Yet, looking at Fig. 5.15b, it is visible that it scores 0 correct 1-1 matches. Figures 5.15c and 5.15d, show that GLMP18 has around 650 queries and has a stable SD. The RPP-attacks have fewer than 10 options for each token.

GLMP19 vs KPT20 vs RPP-Attack



(a) MAE

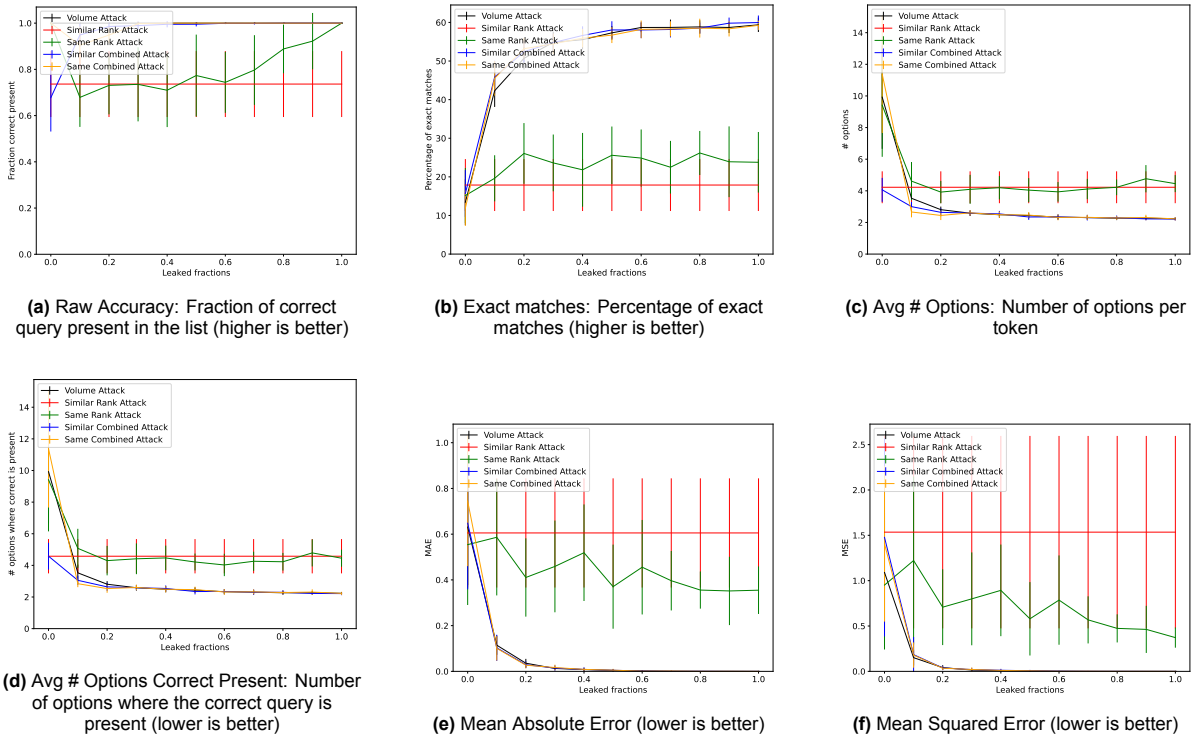


(b) MSE

Figure 5.16: Lucene comparison of own attacks vs GLMP19 vs KPT20 for different fractions of partial data

Fig. 5.16a shows that GLMP19 scores the worst for the MAE even though it has no SD after 10%. The KPT20 attack scored an average MAE of around 3 and has the largest SD whereas our attacks scored below 2.0 and have a smaller SD than KPT20. In Fig. 5.16b, the GLMP19 starts with the highest MSE at 250 after which it stabilizes below the MSE of KPT20 and has a smaller SD than KPT20.

Auxiliary Data



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)

(b) Exact matches: Percentage of exact matches (higher is better)

(c) Avg # Options: Number of options per token

(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

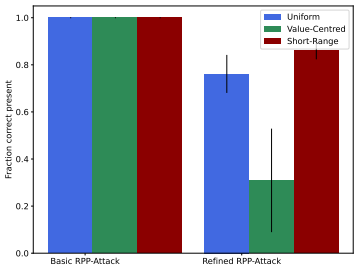
(e) Mean Absolute Error (lower is better)

(f) Mean Squared Error (lower is better)

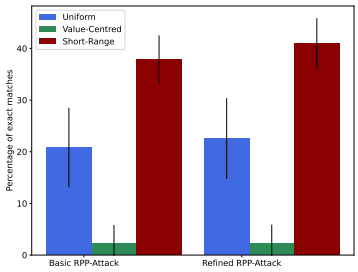
Figure 5.17: Comparisons of metrics for the different auxiliary attacks with the basic variant

In Fig. 5.17, one can find the results for the auxiliary extensions to the basic attack while in Fig. B.2 in appendix B one can find them for the refined variant. Fig. 5.17a shows that all but the same rank and similar rank attacks get close to a raw accuracy of 1.0. A similar trend can be seen in Fig. 5.17b where the maximum is 60 % and the two rank attacks do not go above 30%. The SD is stable for all fractions and has no large spikes. Figures 5.17c and 5.17d visualise that there is a maximum of around 12 options and the rank attacks again have the worst value. The MAE and MSE for both variants in figures 5.17e and 5.17f respectively are below 1.75 for all auxiliary options, with the rank attacks having the worst performance. The SD is also the largest for rank attacks. The refined variant performs differently in the average number of options as visible in Fig. B.2c in appendix B, where the average does not go above 3.4.

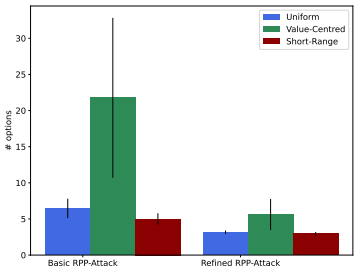
Query Distribution



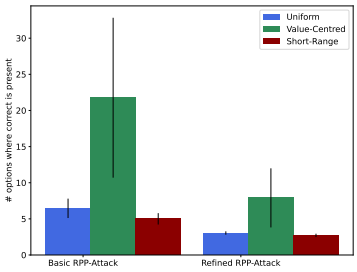
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



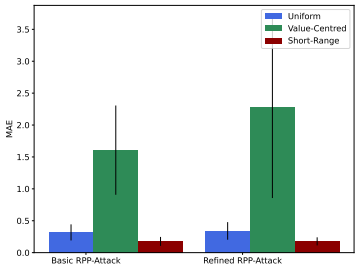
(b) Exact matches: Percentage of exact matches (higher is better)



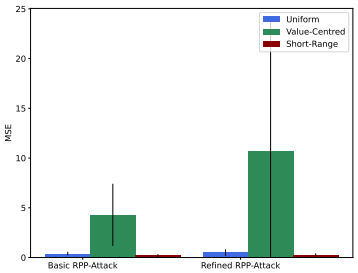
(c) Avg # Options: Number of options per token



(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)



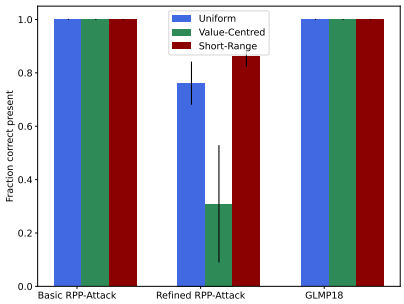
(e) Mean Absolute Error (lower is better)



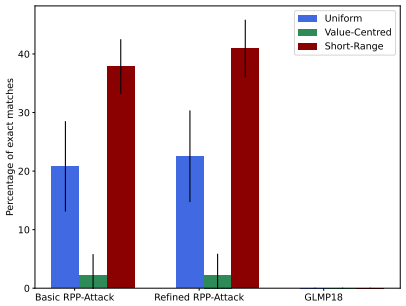
(f) Mean Squared Error (lower is better)

Figure 5.18: Comparisons of metrics for different query distributions for our basic and refined attacks.

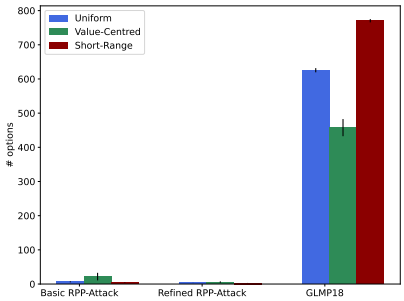
Figure 5.18 displays the results for how the attack performed on different query distributions. The basic attack has the correct query for each token as visible in Fig. 5.18a. For the refined variant, the value-centred distribution has the lowest raw accuracy and the largest SD. Fig. 5.18b demonstrates that the short-range is the best with the highest exact matches and a relatively low SD, followed by the uniform and lastly the value-centred distribution. Looking at the number of options in figures 5.18c and 5.18d, it shows that the value-centred distribution has 2 to 4 times as many options and a larger SD than the others. In the case of the value-centred distribution for the refined attack, while the number of options increases if the correct query is present, so does the SD. For all three distributions, the refined variants do have fewer options. Figures 5.18e and 5.18f make it clear that the average MAE and MSE are the worst for the value-centred distribution, which has the largest SD, and the best for the short-range distribution, which has the lowest SD.



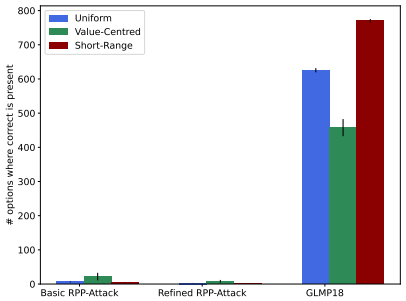
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



(b) Exact matches: Percentage of exact matches (higher is better)



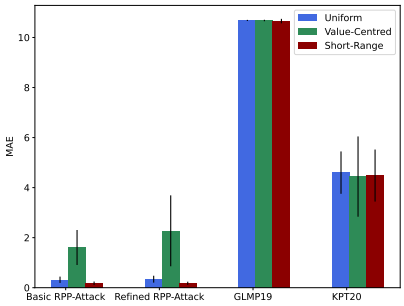
(c) Average number of options for each token



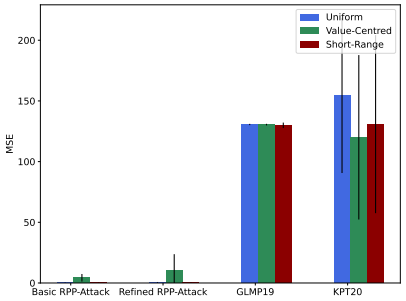
(d) Average number of options for each token

Figure 5.19: Lucene GLMP18 vs own attacks comparisons for different query distributions

For all three distributions, figure 5.19a shows that only the refined variant does not have a raw accuracy of 1.0. For both the basic and refined variants, all distributions have some one-to-one mappings, GLMP18 does not, as shown in Fig. 5.19b. Looking at figures 5.19c and 5.19d, one can observe that for the GLMP18 attack the value-centred distribution has the fewest possible options, yet, it has the largest SD out of the three distributions. The short-range has the most options and the smallest SD. As the raw accuracy is 1.0 for GLMP18 there are no differences in figures 5.19c and 5.19d. The average number of options is more than 15 times as high for the GLMP18 attack compared to the RPP-attack variants.



(a) MAE (lower is better)



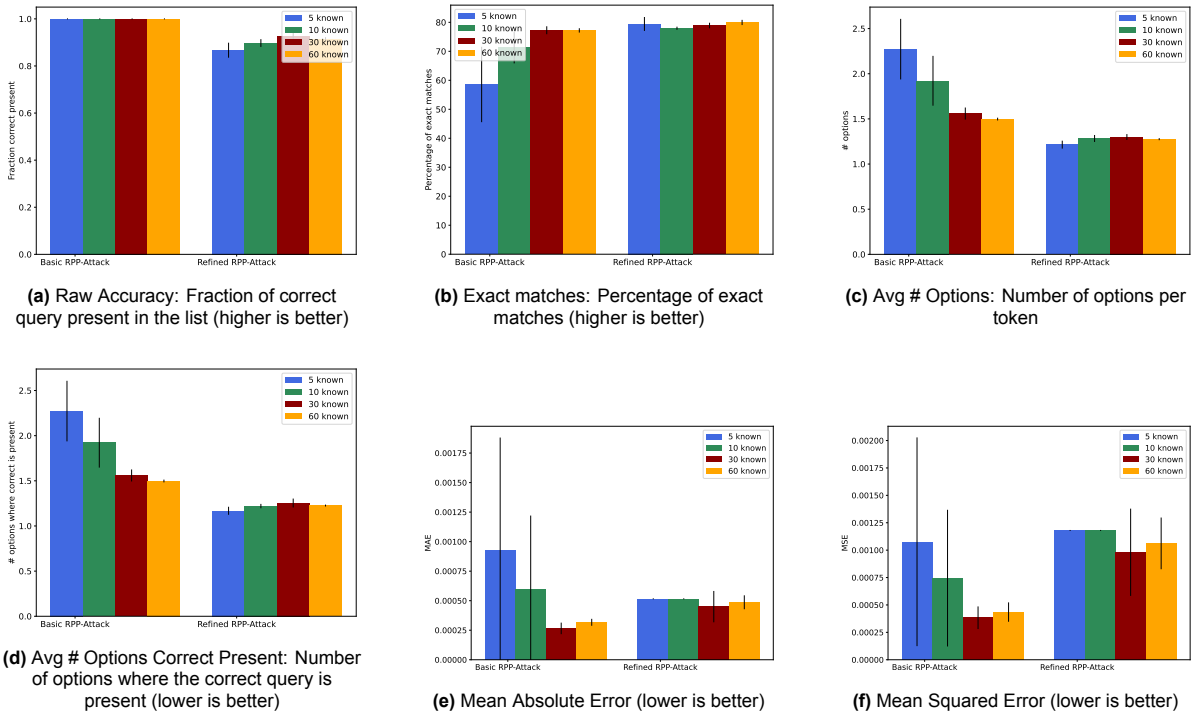
(b) MSE (lower is better)

Figure 5.20: Lucene comparison of own attacks vs GLMP19 vs KPT20 for different query distributions

Looking at the last set of results for the Enron dataset visible in Fig. 5.20a and 5.20b, one can see that GLMP19 has the worst performance. This is the case for all three distributions which score almost the same. The KPT20 attack has the best value for both the average MAE and MSE using the value-centred distribution, while the uniform distribution is the worst. For all three distributions, KPT20 has the largest SD.

5.3.4. UCI Machine Learning Data

Comparison of RPP-Attack



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)

(b) Exact matches: Percentage of exact matches (higher is better)

(c) Avg # Options: Number of options per token

(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

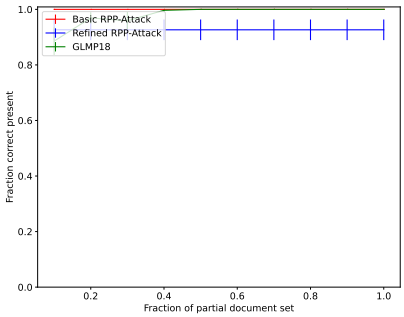
(e) Mean Absolute Error (lower is better)

(f) Mean Squared Error (lower is better)

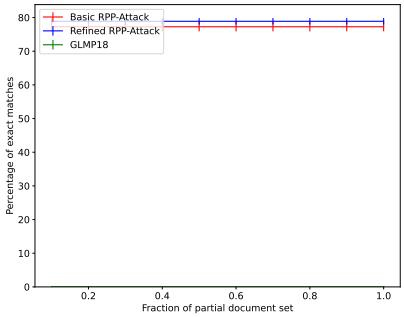
Figure 5.21: Comparisons of metrics for the different number of known queries for our basic and refined attacks.

Fig. 5.21 shows that the raw accuracy is equal to 1 for the basic variant and close to 1 for the refined attack, independent of the known token-query pairs. The SD goes down slightly as the number of pairs increases. The percentage of exact matches is close to 80, for both the basic and refined variants as visible in Fig. 5.21b. It furthermore shows that the SD goes down more for the basic variant than for the refined variant. Increasing the number of known pairs increases the performance for the basic variant, but having more than 30 pairs does not increase it further. In figures 5.21c and 5.21d, one can see that for the basic variant increasing what is known makes the number of options and the SD decrease. This is not the case for the refined variant. Lastly, figures 5.21e and 5.21f show that the average MAE and MSE and their standard deviations decrease for the basic attack as the known pairs increase. The average MAE and MSE for the refined variant do not go down as much when the pairs increase. It's MAE and MSE are higher than that of the basic variant.

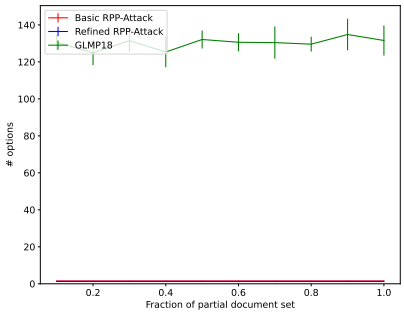
GLMP18 vs RPP-Attack



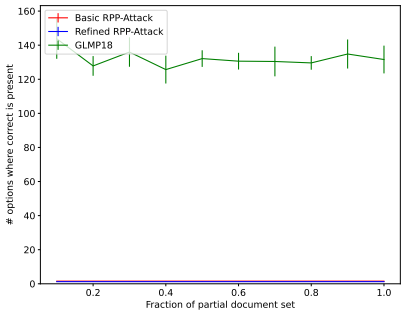
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



(b) Exact matches: Percentage of exact matches (higher is better)



(c) Average number of options for each token



(d) Average number of options for each token

Figure 5.22: UCI GLMP18 vs own attacks comparisons for different fractions of partial data

In Fig. 5.22a one can see that the GLMP18 attack starts with an average raw accuracy of 88 percent. It moves up to 100 percent when half the document set is leaked. Once that point is reached, the SD becomes non-existent. For this dataset, as visible in Fig. 5.22b, the GLMP18 attack is not able to get any one-on-one matches correctly. This is also visible in the average number of queries per token in figures 5.22c and 5.22d. The average does not get below 120. The SD for GLMP18 does not deviate much as the fraction of known documents increases. For our attacks, the average does not get above 3.

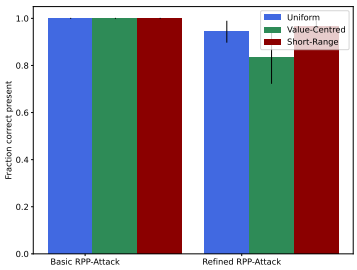
KPT20 vs RPP-Attack

Figures B.3a and B.3b in appendix B tell us that the MAE and MSE for all attacks are low. The KPT20 attack has an MAE of 0.7 and an MSE of 14.

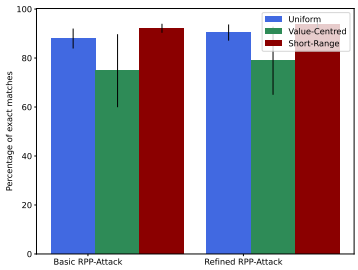
Auxiliary Data

In figures B.4 and B.5 in appendix B, the basic and refined auxiliary variants are shown. From these figures, it is visible that using the auxiliary information increases the exact matches where the same combined attack performs the best. The values are already very good, so the auxiliary information affects the attack with small margins.

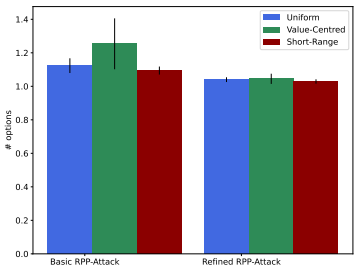
Query Distribution



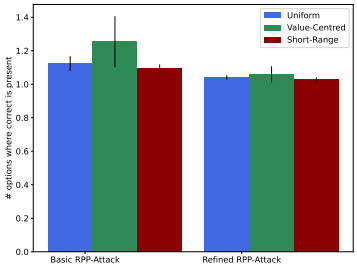
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



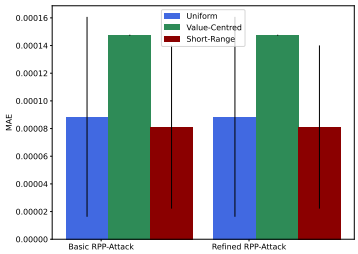
(b) Exact matches: Percentage of exact matches (higher is better)



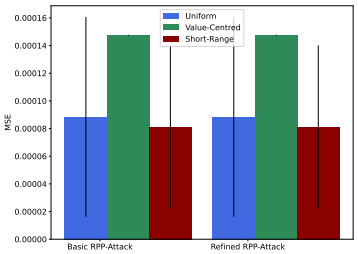
(c) Avg # Options: Number of options per token



(d) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)



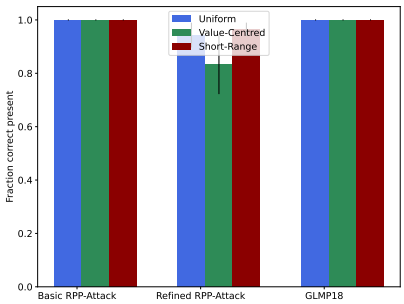
(e) Mean Absolute Error (lower is better)



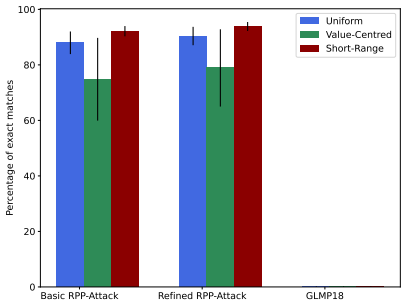
(f) Mean Squared Error (lower is better)

Figure 5.23: Comparisons of metrics for different query distributions for our basic and refined attacks.

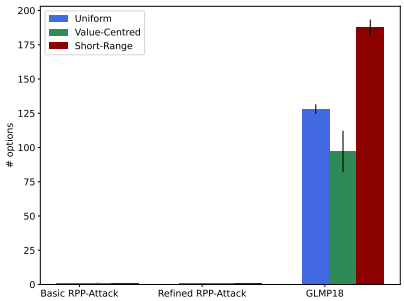
Figure 5.23 displays the results for the performance of the RPP-attack on the UCI dataset for different query distributions. We can see that for all distributions, the basic attack has the correct query for each token. For the refined variant, there is little difference, but the short-range scores the best. This distribution also has the smallest SD. Regarding the number of exact matches, we can see that the short-range is the best, with again, the smallest SD, followed by the uniform and lastly the value-centred distribution which has the largest SD. Figures 5.23c and 5.23d show that the value-centred distribution has more options and the largest SD compared to the others. The MAE and MSE are the highest for the value-centred distribution. Yet there is no SD for this distribution. The uniform distribution is again in the middle and the short-range distribution has the lowest values. Here the MAE and MSE are almost the same for both the basic and the refined variants of the attack. The SD for the uniform and short-range distributions are also very similar.



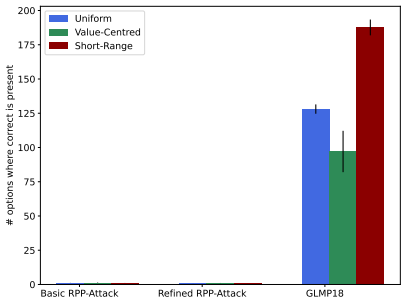
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



(b) Exact matches: Percentage of exact matches (higher is better)



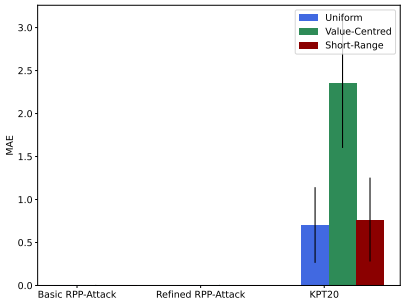
(c) Average number of options for each token



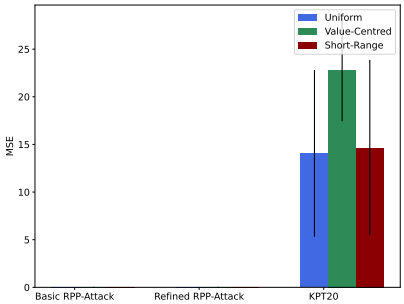
(d) Average number of options for each token

Figure 5.24: UCI GLMP18 vs own attacks comparisons for different query distributions

For all three distributions, figure 5.24a shows that the refined variant does not have the correct query for each token. For the UCI database, the GLMP18 attack has no exact matches, as can be seen in Fig. 5.24b. The number of options for the GLMP18 attack, visible in Fig. 5.24c, is the lowest for the value-centred distribution, yet it has the largest SD, followed by the uniform distribution, and lastly, the short-range has the highest average number of options with 175. On this dataset our attacks outperform GLMP18.



(a) MAE (lower is better)



(b) MSE (lower is better)

Figure 5.25: UCI comparison of own attacks vs KPT20 for different query distributions

Lastly, we can see that for the KPT20 attack the MAE and MSE are the worst for the value-centred distributions, which also has the largest SD for the MAE, and about equal for the other two. The average MAE and MSE have a maximum of 2.5 and 24 respectively. The MAE and MSE for our attacks are not visible because they are so low.

5.4. Run-time

The code for the different attacks and the experiments were written in Python 3.8. We used the PyCharm editor to create and run the experiments. The code was run on an Ubuntu 18.04 system with 16 GB of DDR3 memory, and an I7 4790K at base speeds. The experiments were run sequentially to

ensure that they had the full computing power available.

The run time for our attack on the different datasets can be seen in table 5.2.

Step \ Data set	Basic Enron	Basic Lucene	Basic UCI	Refined Enron	Refined Lucene	Refined UCI
PQ-Tree	37.79	12.63	216.41	37.79	12.63	216.41
Getting results	0.20	0.078	1.97	71.94	27.64	382.56
Total	37.99	12.71	218.38	109.73	40.27	598.97

Table 5.2: Time it took for our attacks on different datasets

From our testing, we also discovered that having a larger domain increases the running time. This was the main reason why we did not go above a domain of 100.

Step \ Data set	GLMP18 Enron	GLMP18 Lucene	GLMP18 UCI
Total	6.55	6.54	2.14

Table 5.3: Time it took for GLMP18 on different datasets

Step \ Data set	GLMP19 Enron	GLMP19 Lucene
PQ-Tree	37.13	11.73
Getting Results	18.44	0.89
Total	55.57	12.62

Table 5.4: Time it took for GLMP19 on different datasets

Step \ Data set	KPT20 Enron	KPT20 Lucene	KPT20 UCI	KPT20 Lucene Unfiltered
Total	5.88	5.80	7.16	171.89

Table 5.5: Time it took for KPT20 on different datasets

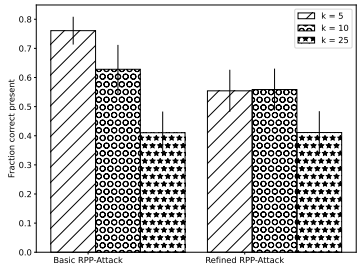
5.5. Countermeasures

We have run the different countermeasures on both the Lucene [1] and Enron dataset [17] for 25 runs and on the UCI dataset for 5 runs. For all the different attacks we chose a uniform query distribution, and for the basic and refined variants of our attack, we chose to have 30 known token-query pairs.

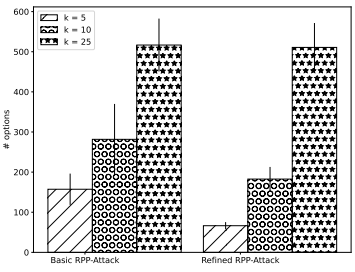
5.5.1. Blocked Queries

For blocked queries, we look at three different values for k, namely $5, 10$ and 25. 5 was chosen because we thought that it would impact the attacks without greatly impacting the performance. 10 was chosen to see if a doubling of the security parameter would result in half the accuracy. The last one, 25, was chosen as a maximum because this has the potential to increase the received range on the server with 50 in the width. As this is half the domain size of the databases, we considered this the maximum before the data overhead becomes too large. We ran the countermeasure on our basic and refined variants, the GLMP18 [28] and GLMP19 [27] attacks and lastly the KPT20 [37] attack. For these countermeasure experiments, the fraction of observed queries was set at 0.15, furthermore, GLMP18 had all data available and GLMP19 again had 80 percent of the server data.

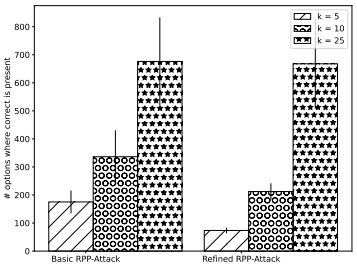
Enron



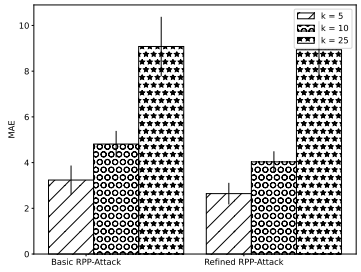
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



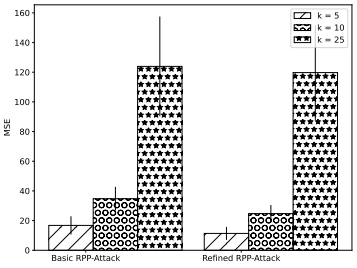
(b) Avg # Options: Number of options per token



(c) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)



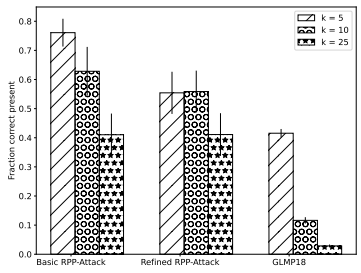
(d) Mean Absolute Error (lower is better)



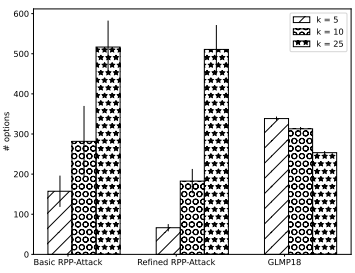
(e) Mean Squared Error (lower is better)

Figure 5.26: Comparisons of metrics for different values of k for own attack on Enron

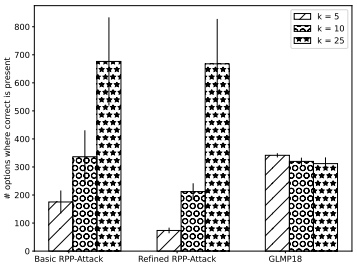
Comparison of RPP-Attack One can see that the raw accuracy for both variants is worse when the countermeasure is in place. Fig. 5.26a shows that the basic variant cannot guarantee that the correct query is present. With a higher k, the SD increases. Furthermore, no attack could make any exact matches. Figures 5.26b and 5.26c show that the average number of options is significantly higher than when there are no countermeasures. The average number of options is even higher when the correct query is present in the list. The MAE and MSE in figures 5.26d and 5.26e respectively, also rose sharply. For all metrics, a higher k increases the SD and decreases the attack performance.



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



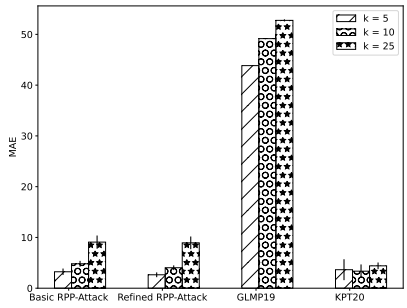
(b) Avg # Options: Number of options per token



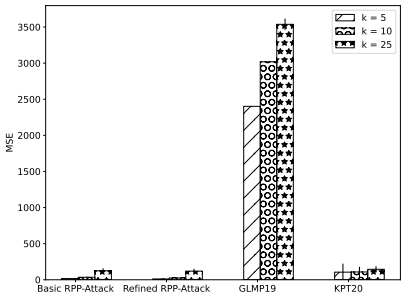
(c) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

Figure 5.27: Comparisons of metrics for different values of k for own attack compared to GLMP18 on Enron

GLMP18 vs RPP-Attack Fig. 5.27a displays that no attack can guarantee that the correct query is present. With a higher k, this effect is stronger. GLMP18 now underperforms to the refined variant. Looking at figures 5.27b and 5.27c, it is visible that the average number of options goes down for GLMP18 contrary to the basic and refined variants. Looking at options alone, GLMP18 outperforms our designed attacks. None of the attacks are able to have any correct one-to-one matches. Furthermore, while the SD has a positive relationship with the security parameter, the SD for GLMP18 does not change much with an increase in k.



(a) Mean Absolute Error (lower is better)

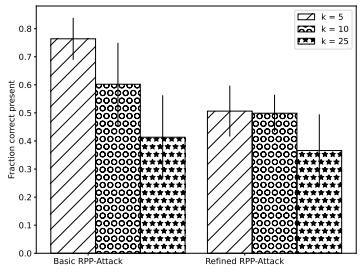


(b) Mean Squared Error (lower is better)

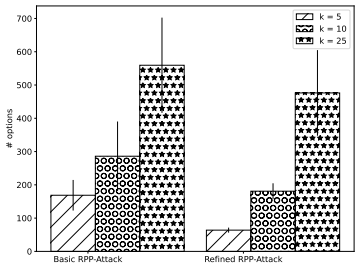
Figure 5.28: Comparisons of metrics for different values of k for own attack compared to GLMP19 and KPT20 on Enron

GLMP19 vs KPT20 vs RPP-Attack Figures 5.28a and 5.28b indicate that all attacks are impacted by the countermeasure. GLMP19 is the most impacted, as its average MAE comes from below 10 to a minimum of 40. Furthermore, for the highest k, KPT20 outperforms our designed attacks. The increase in k from 5 onward has the largest impact on our attacks. Lastly, KPT20 has the largest SD while GLMP19 has the smallest. For all attacks increasing k increases the SD.

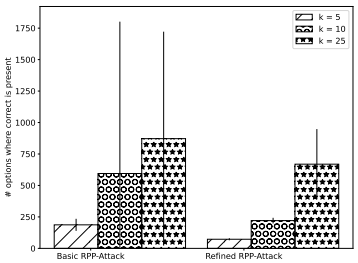
Lucene



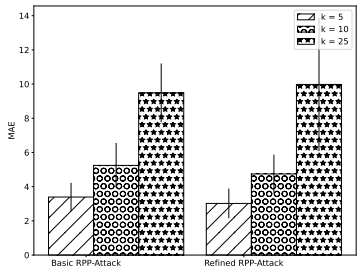
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



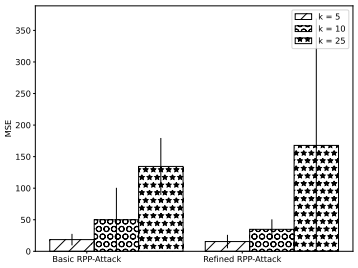
(b) Avg # Options: Number of options per token



(c) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)



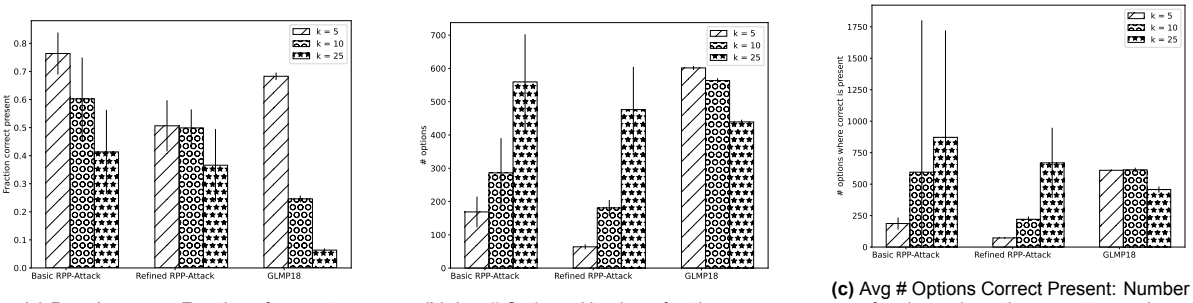
(d) Mean Absolute Error (lower is better)



(e) Mean Squared Error (lower is better)

Figure 5.29: Comparisons of metrics for different values of k for own attack on Lucene

Comparison of RPP-Attack Regarding the Lucene dataset, we see very similar impacts as with the Enron dataset. There are no exact matches and both variants of the attack do not have a raw accuracy of 1 as displayed by Fig. 5.29a. Here too the SD increases with k. The average number of options is higher when the correct query is present for the tokens, as visible in figures 5.29b and 5.29c. Lastly, even though the average MAE and MSE were lower for Lucene without countermeasures compared to Enron, with countermeasures Lucene performed worse. Figures 5.29d and 5.29e show that the RPP-attack performs worse with a higher k. The figure furthermore shows that the SD increases when k increases.



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)

(b) Avg # Options: Number of options per token

(c) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

Figure 5.30: Comparisons of metrics for different values of k for own attack compared to GLMP18 on Lucene

GLMP18 vs RPP-Attack Looking at GLMP18 in Fig. 5.30a, we can see that with $k = 5$ there is still a high raw accuracy. It is higher than the refined variant. The number of options for GLMP18 remains mostly above the basic and refined attack variants. Furthermore, the SD for GLMP18 is smaller than for the RPP-attacks. See figures 5.30b and 5.30c.

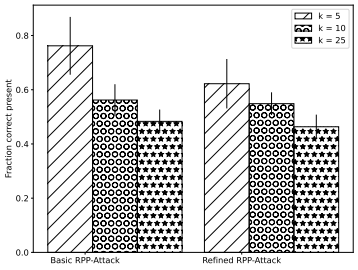
(a) Mean Absolute Error (lower is better)

(b) Mean Squared Error (lower is better)

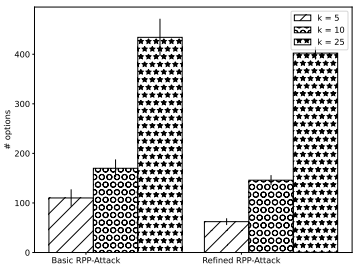
Figure 5.31: Comparisons of metrics for different values of k for own attack compared to GLMP19 and KPT20 on Lucene

GLMP19 vs KPT20 vs RPP-Attack Figures 5.31a and 5.31b indicate that all attacks are impacted by the countermeasure. GLMP19 is impacted the most as it rose significantly more than the rest for the lowest k. For the highest k, KPT20 outperforms our designed attacks, yet, on average, it has a larger SD.

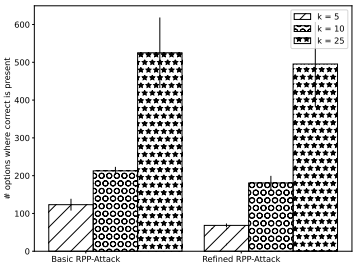
UCI



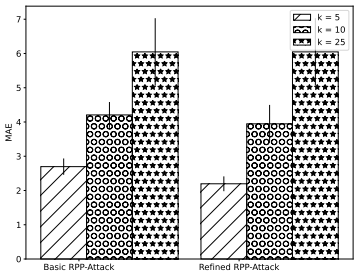
(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



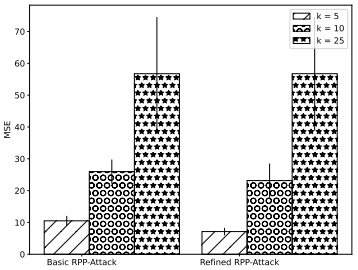
(b) Avg # Options: Number of options per token



(c) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)



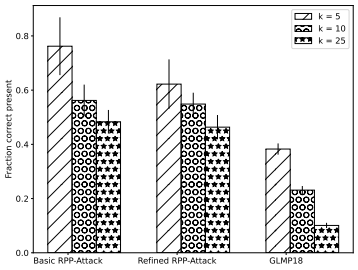
(d) Mean Absolute Error (lower is better)



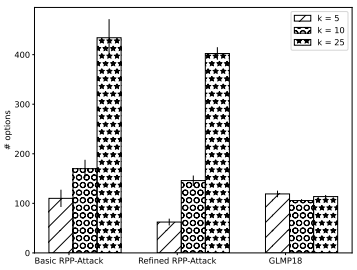
(e) Mean Squared Error (lower is better)

Figure 5.32: Comparisons of metrics for different values of k for own attack on UCI

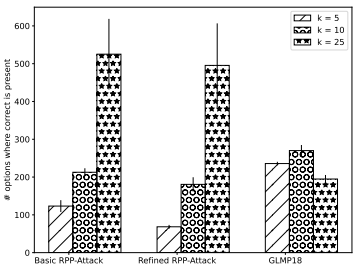
Comparison of RPP-Attack There are no exact matches and both variants of the attack do not have a raw accuracy of 1 as visible in Fig. 5.32a. The number of options is higher when the correct query is present for the tokens, as visible in figures 5.29b and 5.29c. Lastly, figures 5.32d and 5.32e show that while the MAE was lower than 0.1 without countermeasures, the blocked queries set it to a minimum of 2.7. While the SD has a negative relation with the security parameter for the raw accuracy, it has a positive relationship with the other metrics.



(a) Raw Accuracy: Fraction of correct query present in the list (higher is better)



(b) Avg # Options: Number of options per token



(c) Avg # Options Correct Present: Number of options where the correct query is present (lower is better)

Figure 5.33: Comparisons of metrics for different values of k for own attack compared to GLMP18 on UCI

GLMP18 vs RPP-Attack In Fig. 5.33a, one can see that all three attacks are unable to get achieve a raw accuracy of 1.0. Figures 5.33b and 5.33c allow one to see that the options do not go down with a higher k for the GLMP18 attack. Something that did happen with the Enron dataset. Lastly, the SD is smaller for GLMP18 than for the RPP-attacks.

KPT20 vs RPP-Attack Looking at the MAE and MSE we can see that all attacks are impacted by the countermeasure. For all values of k, the KPT20 attack outperforms our attacks. The results are visible in appendix B

5.5.2. Wrap-Around Queries

The wrap-around queries countermeasure was tested by issuing single-length queries to the server. We only ran this for the KPT20 attack because the other attacks require a functioning PQ-Tree to be made, and this is not possible with single-length queries. The GLMP18 also would not work as it would have not known which response volume matches with which actual query as they get split into many single-width queries. The KPT20 attack works, the authors did state that the order has to be available before the attack is run. This means that if the order has to be created using PQ-Trees, this attack would fail too. We display the results for the KPT20 runs below. For this attack, we send all the single queries to be solved and used by the attack.

Countermeasure	MAE Enron	MAE Lucene	MAE UCI	MSE Enron	MSE Lucene	MSE UCI
None	3.22	4.60	0.7	128.25	154.38	14.06
Wrap-Around Queries	3.54	5.04	0.94	134.32	187.88	9.63

Table 5.6: Results of the Wrap-Around Queries countermeasure on KPT20 for the different datasets

KPT20 Looking at how KPT20 got affected, we can see that for almost all metrics the countermeasure resulted in the attack performing worse. The only metric that improved is the MSE for the UCI dataset.

5.5.3. Document Volume Hiding

If the countermeasure ensures that all documents have the same volume k, then all our attacks, except the attack using volume as an auxiliary aspect would still succeed. This is because the other attacks do not use volume. Not being able to use the volume would return the attack to the basic or refined variants. While in the case where the document volumes are padded to the closest multiple of k, the accuracy of the auxiliary volume attack should go down, relative to the security parameter k. The higher the k, the more documents are equal and the less uniqueness there is to compare with. This can partially be mitigated in the attack by increasing the number of combinations. This comes at a cost of run-time, resulting in a weigh-off for the attacker. We decided to not run this countermeasure as it is only applicable to our volume auxiliary attacks and they are not the focus of our work.

6

Discussion

In this section, results will be analyzed and thoroughly discussed. First, the performance of the different attacks on different densities will be discussed. Followed by the separate experiments on the three online datasets. Next, the countermeasures and their effectiveness are discussed. Fourthly, the impact of the different databases and how the RPP-attack might perform on other databases will be looked at. Once this is discussed, the parameter choices in the experiments, and algorithms and how changing them plays a role in the attacks, will be analyzed. Lastly, we will discuss the run-time of the different attacks.

6.1. Density

6.1.1. Comparison of RPP-Attack

Looking at Fig. 5.4a one can see that the basic attack can guarantee that the correct query is always present for all the tokens, but the refined variant cannot. It starts at a fraction of 0.9 for a density of 0.1, and goes down sharply to only having the correct query in the list for 40 percent of the tokens when the density is 0.7. We assume that this is because of the changes associated with the buckets. If the density is low, we have a lot of known token-query pairs that allow us to remove a large number of possible values from the different ranges for the buckets. While, if the density is between half and three-quarters, then the known token-query pairs with empty access patterns might not result in a significant enough refining of the values. This is due to the pairs being unable to eliminate, for example, 20 values, and instead only being able to eliminate 2 values. The higher the density goes after this midpoint, the fewer options are available per bucket at the start of the attack. Furthermore, due to there also being more known token-query pairs with access patterns, the algorithm can assign more exact values to buckets. Lastly, we can see that the standard deviation (SD) is almost the same for each density. This shows that although increasing the density, it does not decrease the SD. Sometimes an attacker will be more successful than other times.

Fig. 5.4b shows that both attack variants have very little to no average exact matches until the density reaches 0.8. Before that, the SD is also non-existent showing that it practically never has an exact match. Once the density is 0.8, the exact matches sharply rise until it reaches 100% of exact matches for a dense database. Regarding the latter, this is to be expected as then all values are present and there is exactly one query that matches each observed access pattern. As the density increases past 0.8, so does the SD until it is a dense dataset. We believe that this happens because there are more chances of exact matches, but due to what is known this fluctuates. We hypothesise that the rise starts at 0.8 because then there are enough documents that are adjacent to each other in value, and have been able to get assigned single values. If, for example, the values $1, 2, 3$ are present and these values are also assigned to individual buckets, then if we observe a token with an access pattern consisting of the documents in the second bucket, we can match the token to a single query. If the density is low, these instances are few and far between.

Figures 5.4c and 5.4d show that density plays a large role in the number of options for each token. Here one can see that the number of possible queries starts with more than 1000 and is already less than

200 for a density of 0.2. The SD also decreases as the average number of options decreases. These sharp decreases are because, with each increase in density, fewer values are possible at the start of the attack for each bucket and there are fewer groups of values that have no documents. Throughout the rest of this thesis, these groups are also called gaps. The methods that assign possible queries for tokens have better starting and end points to work with. Furthermore, the refined attack has fewer options, both overall and when it has the correct query in the list. The fewer options are a result of the method assigning values to buckets at certain stages, which results in a smaller width of options per bucket. Lastly, there is not much difference between figures 5.4c and 5.4d for the refined variant. We hypothesize that this is because when the refined variant assigns a wrong value to a bucket, this value is not too far off, which results in a small decrease of options, but is not enough to change the average significantly.

Lastly, when looking at the MEA and the MSE, one can see that the refined and basic attacks both score similarly. While the refined attack seems to be able to assign better values in the case where the density is low, this is reversed when the database gets denser. Currently, it is unknown exactly why this happens. A hypothesis is that in the case of the refined variant, the attack assigns a value from a shorter list of options per bucket. This means that in sparse databases, the refined RPP-attack is closer to the correct value. Yet when the database is denser, we then, due to assigning shorter ranges, either have the option of choosing a correct value or an inherently wrong value for the document as the correct value might not be in the range due to the refinement. This increases the MEA and MSE for the refined variant. The SD for both variants for both MAE and MSE go down as the density increases. We speculate that this happens because more documents are guaranteed to have the correct value assigned, so there are fewer that we need to guess. This decreases the variation.

6.1.2. GLMP18 vs RPP-Attack

As can be seen from Fig. 5.5a the GLMP18 attack, just like our basic attack, can have the correct query in the list of options for all the density fractions we tested. Yet the attack is unable to find any exact matches, as visible in Fig. 5.5b. This can be explained if we look at the last two graphs. There one can see that the average number of options starts above 2500 and does not get below 1000. The RPP-attack starts below 1500 options and ends at 1 option. The total number of queries to potentially choose from is 5050. Similarly, Grubbs et al. had the GLMP18 attack only make very few, if any, exact matches in their datasets. Furthermore, they showed that if the attack was able to have the correct query in the list, the number of options would often be high [28]. The same is visible on this database. In this specific instance, it is hypothesized that this is due to the close uniformity of the histogram of the database. As the GLMP18 attack bases the matches on the observed volume and the volumes of queries from a known database, and most volumes of queries in our dataset are similar or the same, it ends up selecting a large number of options. The RPP-attack does not look at response volumes of queries, but at the ranges of values per bucket and thus, does not suffer from this limitation. When looking at the standard deviation for the number of options, it can be observed that it follows a similar trend as the RPP-attack and that neither attack has an advantage over the other here.

6.1.3. GLMP19 vs KPT20 vs RPP-Attack

Looking at the MAE and the MSE in figures 5.6a and 5.6b, respectively, it is possible to observe that the GLMP19 has a similar distribution for both. In both instances, the values decrease as the density increases. The KPT20 attack has a different distribution. For KPT20 the MAE goes down sharply in the beginning, after which it plateaus. For the MSE, the KPT20 attack has a horizontal line. These results are similar to those of the original KPT20 authors. They also start around an MAE of 10 when the dataset is at its sparsest, getting close to 0 when the database is dense [37]. The difference between our tests and theirs is the observed number of tokens. They have seen more tokens and thus different results. Looking at the GLMP19 attack and the SD, it is visible that it is unable to confidently assign proper values to the documents. With an average minimum MAE of 10 on a database domain of 100, which is quite large. Looking at the values in our files, it showed that in some runs, the MAE is around 10 or lower while at other times it is around 45 This tells us that the chosen heuristic to get the order does not always work. This is a drawback of GLMP19. The heuristic of the GLMP19 attack looks at how many documents are in front of and behind the middle value and matches this based on the auxiliary dataset [27]. If the dataset on the server does not match the needed requirements, the heuristic will

not work. The KPT20 attack scores significantly better, but the RPP-attack still outperform it.

6.1.4. Overall

Overall we can see that our basic and refined attacks outperform the three attacks for all density fractions we tested. The KPT20 attack comes closest. Furthermore, like all attacks published and the ones looked at here, our attack performs better with a denser database. For GLMP18, KPT20 and KPT20 the consistency of the attacks also improved due to the decreasing SD when density rose. This shows it is important for attacks to be tested on different densities.

6.2. Experiments

Table 6.1 shows how the RPP-attack performed against the GLMP18 attack for the different metrics. Table 6.2 shows this for the comparison between the RPP, GLMP19 and KPT20 attacks. They give a quick overview.

Metric / Attack	Raw Accuracy	Exact Matches	Avg # Options	Avg # Options Correct Present
Basic RPP-Attack	★★★	★★	★★	★★
Refined RPP-Attack	★	★★★	★★★	★★★
GLMP18	★★	★	★	★

Table 6.1: Ranking of attacks on query recovery. More stars is better.

Metric / Attack	Mean Absolute Error	Mean Squared Error
Basic RPP-Attack	★★★★	★★★★
Refined RPP-Attack	★★★	★★★
GLMP19	★	★
KPT20	★★	★★

Table 6.2: Ranking of attacks on document recovery. More stars is better.

6.2.1. Comparison of RPP-Attack

Looking at all three datasets, we can see that the number of known queries plays a large role in the performance of the attack. The figures on raw accuracy show that the basic RPP-attack always has the correct query in the list of options. This effect was expected, as the basic variant was set up to guarantee that this would be the case. For the refined attack, we can see that increasing the number of known token-query pairs increases the number of tokens for which the correct query is in the list. This happens because more queries allow the attack to further narrow down the ranges of values for the buckets. This means that the refinement step of the ”known” data has a smaller probability of being incorrect. While the average raw accuracy improves, the SD remains relatively the same independent of the number of pairs available. This means that knowing more data does not increase the consistency of the attack in that aspect.

When looking at figures 5.7b and 5.14b, it shows that even with five known queries, exact matches can be obtained for both attacks. This increases with the number of known queries. The refined variant also has more exact matches than the basic variant. Both variants getting more exact matches when the number of known token-query pairs goes up is logical. The more known and exact the ranges for buckets will be, the fewer options there are available for each token. This means that there will be more observed tokens which have exact matches. Looking at Fig. 5.21b there seems to be a cap at 80 percent of exact matches. Looking at the SD, it can be observed that this increases as the number of pairs increases. This shows that although the attacks on average have more exact matches, the number of exact matches does vary more. Sometimes an attacker with the same information could be more successful than a different attacker who has the same number of queries.

If we look at the overall average number of options for each token, we observe that it decreases with more known data for the basic variant. The SD also decreases significantly. This again makes

sense due to the aforementioned reason of being able to narrow down ranges for buckets more, which in turn results in fewer options. Fewer options mean less variance and thus lower SD. For the refined variant, this decrease in the overall number of options is not as pronounced.

If we then look at the average number of options where the correct query is present, there is a slightly different picture. For both the Enron and the Lucene databases, there is now a visible decrease in the average number of queries for both variants of the RPP-attack. There we can see that if indeed the correct option is present, having more known queries results in fewer options. This seems to correspond with the increase in exact matches mentioned earlier. The UCI dataset does not seem to have this decrease in the number of options. We believe that this is due to the number of average options already being around 1.25.

Looking at the last two metrics, all three databases have low MAE and MSE. This means that our attack can accurately assign values to documents. Where for the Lucene and Enron databases, it can do so with, on average, no more than an MAE of 2. For the UCI dataset, the MAE does not go above 0.0010. This score is indicative of the attack being able to perfectly assign values to documents, with a few exceptions. Looking at the three datasets, one can see that an increase in the number of known pairs results in a lower average MAE and MSE and SD for both variants of the attack. This is logical as more queries mean a decrease in the width of the possible values for a bucket, or even being able to assign a single value to a bucket and thus less variance is possible in every run as more is known for sure.

Looking at the differences between the refined and basic variants, we can see that the refined variant scored a higher MAE and MSE than its counterpart. As stated earlier, the hypothesis is that the refined variants introduce more errors because, in the process of refining, new ”known” token-query pairs are created and used to assign ranges to buckets. If there is a wrong match, then this can carry over to the other buckets. This means that they might not have the correct value in their range, thus choosing any value for the bucket will be wrong. While with the basic variant, there is always a chance of choosing the correct value for the documents.

Lastly, the smaller fluctuations for the UCI dataset are due to there not being much to improve upon. The MAE and MSE are already very close to 0, meaning almost perfect reconstruction. This is due to the density which results in very few options. Increasing the number of known pairs can improve document reconstruction, but there is always a possibility that one or two buckets might not be contained in the known token-query pairs. If the known token-query pairs could be chosen, an MAE of 0 might be obtained.

Together with what seems to be a ceiling on the number of exact matches, the number of options, and the low MAE and MSE, we believe that there is a maximum on the exact matches and the number of options for tokens. We also believe that our attack can correctly give each bucket its corresponding plaintext value, which should allow for exact matches. But there are a few gaps between values as it is not a dense database. These gaps are visible in, for example, Fig. 5.3a. That means that for some access patterns there is more than one option, but likely not more than 2 or 3. This explains the low number of average options and ceiling for the exact matches. This reasoning extends to the other datasets.

6.2.2. GLMP18 vs RPP-Attack

Looking at the GLMP18 attack and its raw accuracy, one can see that for all three datasets, it reaches 1.0 when the partial document set is set to 50% of the documents or more. At that point the SD also becomes 0 Figures 5.1c, 5.2c and 5.3c show that where the leaked fraction is equal to 0.5, the similarity plateaus. The GLMP18 attack scores better than the refined attack regarding the raw accuracy with a small partial dataset. Yet, it is unable to beat the basic attack, which will always have a raw accuracy of 1.0. Furthermore, while the RPP-attacks can make exact matches, the GLMP18 attack cannot. The original authors also did not have a dataset where GLMP18 was able to make exact matches [28].

One can see that throughout the different datasets, the RPP-attacks have significantly fewer options. When our attack has fewer options, so does GLMP18. The differences between the attacks are large. For example, in the Lucene dataset, there are 10 and 5 options for the basic and refined variants respectively, while GLMP18 has more than 600 options. This clearly shows that our attack outperforms

the GLMP18 attack when looking at the certainty of an attacker. Even though the GLMP18 attack scores worse, it only has 7, 12 and 4.54 percent of the queries as options for the Enron, Lucene and UCI datasets respectively. An attacker can then already have some idea of what is being queried. This shows that indeed ”simple query reconstruction attacks can reveal fine-grained information about queries and damage privacy” [28]. The results also seem to indicate that increasing the similarity of the auxiliary database does not necessarily increase the performance of this attack. This is consistent with the findings of Grubbs et al. [28]. Lastly, the GLMP18 attack has, like the RPP-attacks, a relatively low SD for the number of options. This shows that all attacks are consistent in the number of options they return for each run. This consistency allows an attacker to often succeed.

Our attacks outperform the GLMP18 attack because, first off, the RPP-attack needs to see more tokens. GLMP18 can be run on any single observed token. Furthermore, the run-time of our attack is higher and requires significantly more computations.

6.2.3. GLMP19 vs KPT20 vs RPP-Attack

Looking at the differences between the GLMP19, KPT20 and our attacks, one can see that in the Lucene and Enron datasets there is a clear order. The GLMP19 attack has the worst MAE and MSE, followed by the KPT20 and lastly our attacks. For the UCI dataset, the GLMP19 was not able to be completed because of the run-time of the algorithm. An interesting observation is that the MAE and MSE for GLMP19 do not seem to go down with increasing partial documents available for the Enron dataset, but do slightly for the Lucene dataset. Looking at the results, we believe that this is due to the heuristic sometimes failing if there is not enough information. This can also be observed in the SD which is present for the Lucene dataset but not for the Enron dataset. The heuristic failing is a problem because then the attacker would make wrong estimations. Furthermore, it results in a relatively bad performance compared to the other three attacks. These problems were also mentioned by Grubbs et al. [27]. If it would always orient correctly, GLMP19 would be a better attack. As visible in the low to non-existent SD, when GLMP19 does have enough data and can orient correctly, it always orients correctly.

KPT20 and our attacks do not have that problem. The RPP-attacks were able to orient the subset correctly in all experiments. KPT20 scores an MAE of less than 5 for both Enron and Lucene and 0.7 for the UCI dataset. Yet our two attacks still outperform the KPT20 attack on all datasets. Furthermore, the SD of the RPP-attacks is smaller, showing that the RPP-attacks are more consistent. This is likely because the RPP-attacks have extra knowledge through the known-query token pairs, whereas the KPT20 attack requires no auxiliary knowledge. Not requiring this knowledge makes KPT20 stronger in that regard. Furthermore, if more tokens would be observed, then KPT20 would be able to make even better estimations and get a lower MAE and MSE than it currently has [37]. Yet our attack would not necessarily benefit from more queries. Once it had the correct number of buckets, observing more tokens does not help.

A large drawback of the KPT20 attack is that it requires only one document per value to be present to work effectively. If there are duplicate values in the database, then the algorithm needs to be modified to solve it correctly. These modifications result in a larger and more difficult equation to solve, and a longer run-time. Due to the longer run-time, we were unable to see how the attack performed if we used all the documents from the datasets on the UCI and Enron datasets. This problem was also encountered by Kamara et al. [33]. For KPT20 attack to start, it requires having the correct order. Kornaropoulos et al. suggest using the PQ-Tree technique used in the GLMP19 attack to get the order [37]. The KPT20 would then suffer from the same problems as GLMP19. Thus the KPT20 works well if an attacker has no auxiliary information, the correct order, and can filter out unique values. The latter would not be possible in $L1$ schemes.

6.2.4. Auxiliary Data

The performance of the auxiliary attacks will be discussed. The attacks are divided into three groups. The volume auxiliary attack, the rank auxiliary attack and the combined attacks.

Volume
The volume auxiliary attack is capable of maintaining the guarantee that the basic RPP-attack always has the correct query in the list. For the refined variant, it improves the raw accuracy significantly, compared to not having extra information, without needing more than 40 percent of the documents to be leaked. This works best on the Lucene and Enron datasets and less on the UCI dataset. This is because of the distribution of the document volumes. For Enron and Lucene emails we were able to use the actual document volumes, but not for the UCI set. In our reproduction of the attacks by Blackstone et al. [5], we discovered that the distribution of volumes plays a large role in these kinds of volume-based auxiliary attacks. The volume auxiliary attack also increased the exact matches from 5 and 10 percent for the Enron and Lucene datasets respectively, to 60 percent for both. For both datasets, the auxiliary attack also lowered the number of options down to around 2. To achieve this, it required a maximum of half of the document set.

There also seems to be a limit on how low the MAE and MSE can get. This limit is reached at the same time when there is a limit for the options and the exact matches for both Enron and Apache datasets. Our idea is that this limit exists due to the distribution of the datasets, as mentioned earlier. We will go further into this in section 6.4.

These results show that the volume auxiliary information is capable of greatly increasing the performance of the attack without needing much leakage, as long as the distribution of the volume is diverse.

Rank
When looking at the rank auxiliary attack, there are two variants, one that uses a partial document set, and one that uses a similar document set. Looking at sub-graph a for 5.10a,B.1a,5.17a,B.2a,B.4a,B.5a, one can see that across the three datasets, the basic attack cannot guarantee that the correct query is present. It first goes down, and then up to 1.0 after more of the partial document set is known. The partial and similar document sets for Lucene and UCI are more useful than the ones for Enron. The hit in raw accuracy is smaller for those.

One can see that the increase in exact matches for the partial document set follows an almost identical shape for both the basic and refined variants. This is the same for the similar rank attack for the Lucene and UCI datasets. For both these two datasets, the MAE and MSE go down using this auxiliary information. The lowering of the MAE and MSE explains the increase in exact matches because the attack can assign more exact ranges to the buckets of documents. Looking at all datasets, it is visible that the MAE, MSE and the number of options do not go down as low as for the other auxiliary attacks. Other auxiliary attacks also score a higher percentage of exact matches.

Looking at the similar rank attack on the Enron dataset, it scores quite poorly compared to other similar dataset attacks. This shows that the similar document set for Enron is not good enough to use. This was unexpected as the KS similarities of both the Enron and Lucene datasets are low, 0.101010 and 0.1 respectively. We had thus expected the attack performance of this auxiliary variant on the Enron database to be similar to the counterpart on the Lucene database. It shows that the KS similarity value does not necessarily define how well the attack performs. Grubbs et al. [28] had the same findings.

Combined
Looking at the combined attack, it is clear that the partial document set outperforms the similar document set attack. This is expected as the KS similarity of the partial datasets are lower. Furthermore, one can see it follows largely the same results as the volume-based auxiliary attack. The difference is that the added rank leakage slightly increases the performance. The combined attack reaches a plateau of exact matches quicker and ends up slightly higher in the exact matches. Furthermore, the MEA and MSE are lower than for the volume attack on its own. This shows that adding rank leakage to the volume leakage increases the performance of the attack.

Overall we can see that the auxiliary attacks, even with only having a 10 percent of the data available increases the performance of the attack. For the Lucene dataset, we can see that it jumps from 20 to 60 percent of exact matches for some of the auxiliary attacks. Furthermore, the MAE and MSE go down. This shows that if we have the auxiliary information available, the RPP-attacks become even more useful. Compared to GLMP18 and GLMP19, our attack outperforms them even more if the same

amount of data is available. Furthermore, on average the SD is lowest for the volume-based auxiliary attack, followed by the rank auxiliary attacks. This is because the volume-based attack will not make any mistakes if the correct number of buckets are present. The rank-based attack performs better or worse based on the available data. As the combined has both aspects the SD is also in the middle

6.2.5. Query Distribution Comparison

This next section will focus on query distribution and what kind of impact it has. First, the two variants of the RPP-attack will be looked at, and afterwards, the GLMP18 attack compared to the RPP-attack. Lastly, we will look at the GLMP19, and KPT20 attacks and compare their performance with the performance of our attack.

Comparison of RPP-Attack

Looking at how the query distribution affects our attacks, one can observe that the attack performs the best on the short-range, followed by the uniform and lastly the value-centred distribution. The SD is also on average the smallest for the short-range distribution and the largest for the value-centred distribution. On all datasets, the short-range distribution has more exact matches, has a higher raw accuracy, and has the lowest MAE and MSE. The difference between the short-range distribution and the value-centred distribution is rather large. On the Enron and Lucene datasets, the uniform and short-range distributions get significantly more exact matches than the value-centred distribution. Furthermore, the MAE and MSE are, at least, twice as high for the value-centred distribution. Lastly, the short-range distribution results in the most consistent attack performance, resulting in an attack always being good, while for the value-centred distribution, an attacker can sometimes get results that differ a lot between runs.

Looking at the distributions, we hypothesise that the short-range distribution is the best, partly because it practically ensures that the PQ-tree has the exact correct number of buckets. This is because short-range queries make it easier for the PQ-tree to form correctly. The tree gets formed correctly because the queries have directly adjacent overlaps, which allows the PRR-attack to put the document identifiers in the correct node. The value-centred distribution on the other hand centres around specific values. If one goes further away from that value, the probability is higher that a group of documents is only observed in one query and not in others. Resulting in them being put in one bucket instead of multiple. This means that there are fewer buckets, thus increasing the initial ranges for each bucket, and some documents are guaranteed to be assigned the wrong value. This did not happen in our runs, but this is something that needs to be taken into account.

Furthermore, having known short-range token-query pairs likely allows the algorithm to assign exact, or almost exact values to buckets. This is because the probability of having queries of length 1 or 2 is high. While with the value-centred distributions, the RPP-attacks likely assign the exact value for the target value, while document buckets which have values further away will have a wider possible range. This is because fewer queries hit them and thus fewer queries are available that can refine the ranges of those buckets.

The uniform distribution is second, which we believe is due to several reasons. Firstly, this distribution ensures that most documents get placed in their correct buckets. Secondly, the known pairs allow for better refinement than the pairs from the value-centred distributions as they are more of a mix of long and short queries. Yet they will not be able to narrow the ranges down as effectively as the short-range distributions because the list of queries contains longer queries. These longer queries will more likely span empty gaps meaning that the ranges are narrowed less.

GLMP18 vs RPP-Attack

Comparing the RPP-attacks to the GLMP18 attack, it is visible that GLMP18, just like the basic variant, has all the correct queries for all the query distributions. For the GLMP18 attack, the value-centred distribution has the fewest options for each token, followed by the uniform, and lastly the short-range distribution. Which is the inverse order compared to the RPP-attacks. Furthermore, while the value-centred distribution is the best for GLMP18, it also has the largest SD. Yet as the SD is small, it still outperforms the other two distributions by a large margin.

GLMP18 chooses possible queries for a token based on the observed volume and the expected volume of queries using the auxiliary distribution. In the case of the short-range distribution, there are a lot of single-domain queries, if there are many values that have a similar number of documents, the options quickly rise. Furthermore, if the range is a bit wider there could only be a slight increase in the volume for these values. This means that a lot of queries can have similar enough volume sizes to the observed response volume. We think that the value-centred distribution performs the best because the number of documents of at least 1 chosen value will always be present in the response volume of the observed query. This means that any possible query that has fewer documents in its response will most likely be eliminated which can remove a large number of possible options. We hypothesise that the uniform distribution is neither the best nor the worst because it has a higher chance of hitting a larger amount of longer-range queries which have a larger response pattern for which there are fewer possible queries.

GLMP19 vs KPT20 vs RPP-Attack
Looking at the GLMP19 attack we can see that the different distributions do not make much difference for the average MAE and the MSE as the SD is almost 0. This is because the GLMP19 attack looks at the ranks of values and is not heavily based on any query distribution. The only aspect that takes the distribution into account, for this attack, is the calculation of how many documents it was unable to put into the PQ-tree [27]. Yet this is of no relevance in our experiments because all documents were always observed and no documents had to be thrown out.

For KPT20, the results align with the findings by Kornaropoulos et al. [37]. They found that their attack performs best on the uniform distribution with the short-range distribution only lacking somewhat. They also saw that the value-centred distribution was an order of magnitude worse [37]. We do not get such a large distinction. This might be due to the number of observed tokens. We can also see that KPT20 struggles with the value-centred distribution when looking at the SD. This problem exists because the distribution focuses on specific values and thus does not explore the universe of access patterns as much compared to short-range or uniform distribution [37].

These results show that our attack outperforms the other attacks for any given query distribution. Yet looking at the value-centred distribution, improvements can be made because this is a limitation in our attack.

6.3. Countermeasures

6.3.1. Blocked Queries

Comparison of RPP-Attack
When looking at the blocked queries countermeasure, it can be observed across all three datasets that the RPP-attacks cannot guarantee that the correct query is present in the list for the tokens. With an increase in the value of k, the raw accuracy goes down. The attacks also do not have any exact matches for any dataset anymore. Figures 5.26b, 5.29b and 5.32b show that with countermeasures there are significantly more options for each token. It starts at 50 options and gets as high as almost 550 for the highest k. Looking at the average number of options when the correct query is present, the increase is even larger.

These effects are because of how the countermeasure works. As the query send gets transformed into a longer query, the access pattern becomes that of the longer query. This means that if the actual query was $[3, 5]$ and $k = 5$, the query would become $[0, 10]$. If values lower than 3 and between 6 and 10 exist in the database, the RPP-attack will not guess $[3, 5]$ because that access pattern would be too short. The significant increase in the number of options can also be explained. As there are significantly fewer distinct queries, the number of buckets also goes down. This effect is strengthened by having a higher k. Due to having fewer buckets, the range per bucket increases and thus the number of possible options.

The result of fewer buckets and there being documents wrongly put in the same bucket is visible in the MAE and MSE graphs for the three datasets. For all three, the MAE and MSE rose compared to not having countermeasures in place. The order of the datasets has stayed consistent. The UCI

dataset still has the lowest values and Lucene has the highest. The MAE and MSE increase when k increases. As mentioned, this is due to having more documents wrongly put in the same bucket.

Having the blocked queries countermeasure means that our attack performance is worsened. It even negates having exact matches on an almost dense dataset with a k as small as 5. This k could be as low as 2. Furthermore, if we look at the SD, we can observe that it increases when k increases. This means that as it results in a worse-performing attack, it also results in the attack being less consistent and thus of lesser use to an attacker. These results show that although the attack works well in normal systems, it performs significantly worse if protection methods are in place. Showing the importance of countermeasures.

GLMP18 vs RPP-Attack
Looking at the results for the GLMP18 under the countermeasures, one can see that this attack is now also not able to guarantee that the correct query is present in the list. The decrease in the raw accuracy is significantly worse for the GLMP18 attack than for the basic and refined RPP-attacks. Our attack's lowest raw accuracy value is still higher than GLMP18's highest. Furthermore, the increase in k also has a worse effect on GLMP18, bringing the raw accuracy down to 0.1. We believe that this effect is caused by the increase in the number of documents for a token. With an increase in k, there are fewer and fewer queries that come close enough in response volume. With the highest k and thus the highest width, there are only very few queries that would match. Where our attacks show an increase in the average number of options, there is a decrease in the number of options for the GLMP18 attack. This effect occurs because of the aforementioned effect of this countermeasure. Fewer queries can match with a possible token so fewer options are in the list for GLMP18. The decrease in options here is thus not very beneficial. An advantage this attack has over the RPP-attacks is that the SD remains very constant for each k. This means that while the GLMP18 performs worse with each k, the attacker knows what he can exact for each run. Something it can not with the RRP-attacks.

Our attack performs better than the GLMP18 attack if these countermeasures are in place. Even though the RPP-attack has an increase in the number of options, it nonetheless has a higher raw accuracy, making the attack still more useful for attackers. The only case where GLMP18 outperforms is when an attacker knows that the server has a high k value in place and the attacker is only interested in token-query pairs that have a wide range. Then the fewer options from GLMP18 can be beneficial.

GLMP19 vs KPT20 vs RPP-Attack
Looking at the last two attacks, GLMP19 and KPT20 one can see that all attacks take a hit. Some more than others. GLMP19 has a significantly higher MAE and MSE than when it has no countermeasures. When there are no countermeasures the MAE is around 10. We can also see that with a higher k, the MAE and MSE go up. This is because there are fewer buckets for GLMP19 just as with the RPP-attacks. As is visible, the RPP-attacks also suffer from this effect but still have a smaller MAE. Furthermore, there is no almost no SD for GLMP19, meaning that although it performs worse due to the number of buckets, it is the only factor at play and not the randomness in the data available for the attacker.

The KPT20 attack takes a small performance hit. Here there is also a decrease in performance and a higher MAE and MSE if k goes up. The KPT20 attack has a performance hit because it estimates how often a specific access pattern occurs. Due to the countermeasure, certain access patterns are observed more often than they should have been observed. Yet, as is visible, the estimation is still close to the truth. This countermeasure still makes KPT20 perform extremely well.

Looking at the comparisons here, we can say that our attack outperforms the GLMP19 attack under these circumstances. The KPT20 attack under these countermeasures performs similarly to or even better than our attack as it is less dependent on the access pattern and is thus the best out of the three for this countermeasure.

6.3.2. Cost of Countermeasure
Although the countermeasure is efficient at negating parts of the different attacks, it does come at a cost. This cost is in the overhead of the documents sent over the network and the resources needed to remove the false positives before showing the result to the end user. If the security parameter becomes

too large, then larger sections of the database get retrieved. One can then wonder if the overhead and extra time needed to use this countermeasure is something that the average consumer is willing to accept or that they would rather have a less secure, but more user-friendly program.

6.3.3. Wrap-Around Queries

KPT20

The KPT20 attack does suffer somewhat from this countermeasure as can be seen in table 5.6. The MAE and MSE are higher across all datasets. There are only as many single-width queries as the size of the domain. If the domain is 100, normally 757 tokens would have been available to get the results, now only 100. This means that the KPT20 loses aspects it can learn from when it only observes single-length queries. Yet this is the only attack that is still able to get results as long as the order of documents is known beforehand. If this is not the case, then no attack would succeed with this countermeasure. Which was also the goal of the countermeasure [42].

Cost of Countermeasure

As stated, the cost for ”Blocked queries” is quite high, and the cost of the Wrap-Around countermeasure is even higher. The cost is higher because more information has to be sent over the network and more calculations have to be done. As a single issued token gets added to a pool of other tokens, the total number of queries increases. This pool of queries furthermore, gets converted to singleton queries which are then handled by the server. This means that the server has significantly more tokens to process. As the pool has to be handled, more documents are sent over the network back to the client. In the worst-case scenario, the entire database can be sent back to the user. Next, the software on the client has to filter out the false positives. All of this results in more resources needed on the client, server and network. For the average consumer, this increase in resources is likely too much compared to the security they get in return.

6.3.4. Document Volume Hiding

As mentioned document volume hiding only affects our volume auxiliary attack. This means that this countermeasure does not impact the basic and refined RPP-attacks. Less improvement can be obtained, but some improvement could still be obtained using the rank auxiliary attack.

6.3.5. Other Countermeasures

We have previously discussed several countermeasures that could be implemented to make attacks less efficient. From these results, we saw that certain aspects of our and other attacks can be blocked or mitigated. There are two aspects of our attack that can be impacted. The first is the creation of a proper PQ-Tree. The second is to have our attack assign good ranges to buckets.

The first aspect is impacted by the Wrap-Around and Blocked queries countermeasures. The first blocks the creation of the PQ-Tree in its entirety, nullifying the RPP-attack. The blocked queries aspect partially achieves this by having fewer queries. Fewer queries mean fewer distinct options to create the appropriate number of buckets of documents. Another option that was not tested, would be to have the countermeasure return the correct documents plus documents with random values. This countermeasure was described by Cash et al. [13]. If this occurs, then the PQ-Tree would try to reduce the tree given the access pattern and run into problems. This is because the algorithm finds it cannot keep the constraints and thus fails the attack. It does allow an attacker to know something is off.

The other aspect was achieved by the blocked queries. This was as it made sure that due to the width of a query becoming larger, smaller range queries were not added to the list of options. This shows us that such aspects are achieved when results span more than was expected. A similar negative impact could happen when the issued tokens would be transformed into smaller tokens. It would introduce false negatives but then the possibly correct longer queries would not be in the list of options.

6.4. Datasets

Throughout the experiments, multiple datasets were used to see how the attacks performs. More datasets are available online and these could have different results. In this section, how datasets impact the performance of attacks and what an attacker needs to be aware of will be discussed.

The Enron and the Lucene dataset, have high densities. 0.88 and 0.85 respectively. Yet, the results are not what was initially expected. We expected more exact matches for the Enron compared to the Lucene dataset. Because on average, a higher density results in a better-performing attack as could be observed in the experiments on the density dataset.

This means that other aspects impact performance than just the density. As mentioned previously, a possible cause could be the distribution of the database. The distribution of the database indicates where the document values are not present. These gaps, even on a dataset with a higher density, can result in a higher average number of options. When a dataset has, for example, one continuous gap instead of multiple smaller gaps, there are only a few tokens which will get matched with a large number of options, while if there are multiple smaller gaps, there are more tokens which have somewhat fewer query options.

Secondly, and this is present in all RPP-attack variants, as the MAE goes down, the number of options goes down and more, and the exact matches rise to a certain plateau for the Enron and the Lucene datasets. Furthermore, as seen with the UCI dataset, even though the dataset is rather dense, there are a lot of documents and the MAE is 0 except for a few instances, the RPP-attack is not able to assign 1 query to each token. This again is due to the gaps in the histogram of the dataset. Due to those gaps, some tokens will always have at least two or more options.

Databases with a low density (below 0.5), such as our density test database, have very limited or no exact matches and the number of options is significantly higher than on high-density databases. Even for the used density datasets, the MAE is relatively low with a maximum of 4 at the lowest densities. This means that although the RPP-attacks assign document values relatively well, it has trouble finding exact matches. This again is due to the aforementioned gaps in distribution. Yet if there is a low-density dataset but the gaps are for example two continuous gaps, then the RPP-attack could still be able to get exact matches as there are then sequential document values.

This shows that our attack performs differently based on several aspects. First off, the density. The lower the density, the more options per token on average due to the gaps and the wider range of possible values for each bucket that gets assigned at the start. This results in a higher MAE and MSE. With a higher density, we start with fewer possible values per bucket, and we have a lower MAE and MSE. Due to the higher density there more sequential values are present and as the attack has a low MAE and MSE, these document values are known. This allows the attack to assign exact matches to tokens. The distribution also affects how well the known token-query pairs can assign exact values. If there are gaps in the values of the known token-query pair, then the RPP-attack can only assign ranges compared to when the known token-query has no gaps in the document values it encompasses.

To conclude, the accuracy of our attack depends on the density and distribution of the dataset This means that two datasets with equal density and an equal number of documents, but with a different distribution of values will result in two different sets of results.

6.5. Parameter Choices

Throughout the experiments, different parameters were used that impacted the performance of the different attacks. In this section, these parameters and how changing these could impact performance will be looked at.

6.5.1. Number of Known Token-Query Pairs

As discussed, the different number of known queries can make a large impact on the performance of the attack. The values $5, 10, 30, 60$ of known queries were chosen to see the difference. 5 is a relatively low number and allowed us to see how well the attack performed with a low number of known queries. 10 showed how much difference doubling from this point made. 30 allowed us to see the effect of

increasing it further and lastly 60 would show if increasing it more, had a significant enough impact on the performance.

The results showed that increasing the known token-query pairs is worth it from an attack performance standard. Using these 4 numbers, the only metric where a clear ceiling is already observed is in the number of options for each token. Yet in the MAE, MSE and exact matches, there is enough of a difference to expect that increasing the number of known pairs, even more, will result in even better results. There will be a maximum usable number of known queries. This is because, at that point, having more does not allow the attack to further narrow down the exact values of the different buckets. Each bucket then has the correct value assigned and the MAE and MSE will already be 0.

Lowering the number of known tokens will result in a worse-performing attack. This as fewer known token-query pairs means that the attack has less to use. At a minimum, the attack requires having two known token-query pairs with a non-empty access pattern of which a minimum of 1 document is not in both access patterns. These two queries are needed to retrieve the order of the gotten buckets from the PQ-Tree.

6.5.2. Observed Number of Tokens

In our experiments, the attacks observe 15% of all the possible queries that a user can issue. This value was chosen as we believed that it was realistic for an attacker to observe this portion of the queries compared to observing all. Observing all of the queries means that an attacker would have to observe a lot of duplicates according to the coupon collector problem [34]. This portion was also chosen to show that the RPP-attack works without needing to have observed all tokens.

Increasing the observed queries will not necessarily result in a better-performing attack. This is because, with 15%, the attack can create the correct number of buckets. If the other parameters would be optimal, the RPP-attack would be able to get the maximum of exact matches. If, on the other hand, our attack would observe fewer queries, a point will be reached where documents with different values will be grouped in the same bucket. If this happens, the performance of the attack will go down as documents will be miss classified.

6.5.3. Query Distribution

There is a difference in how all of the attacks are performed on the different distributions. In the experiment that compares the attacks based on leakage and densities, uniform distribution was used. This distribution was chosen as this is the most used distribution in the literature. Most of the attacks are either based on this distribution or require knowing which distribution is in use. Looking at the query distribution results, it is also visible that the uniform distribution is often not the best nor the worst, which makes it a good comparison metric as then neither attack has an inherent advantage over another attack due to the query distribution.

6.5.4. Leaked Data

For the GLMP18 attack, we set the leaked data at 100% and for GLMP19 it was set to 80%. These values were chosen such that these attacks would have an advantage in what information they had available to successfully execute the attack and get the best results from an auxiliary data standpoint.

With these advantages, our attack still outperformed the GLMP18 and GLMP19 attacks. Furthermore, increasing the leaked fraction for the GLMP19 attack would not give the attack much or any real improvement as is visible in figures 5.9a, 5.9b,5.6a,5.16b. Increasing this fraction did not lower the MAE or MSE. Lowering the fraction also does not have a significant effect on GLMP19. Thus we can choose any value for this parameter without changing the results significantly.

Looking at GLMP18, lowering the data available for this attack will matter somewhat. If the fraction of available documents for the attacker gets below 0.5, then the GLMP18 attack will not be able to properly find the correct query for each token because there is not enough information available.

6.5.5. Parameters GLMP18 and GLMP19

Both GLMP18 and GLMP19 have parameters that impact the performance of the attacks. GLMP18 has a performance parameter ϵ. This parameter can be used to tune the number of options per token. As an effect, it also impacts the raw accuracy. The original authors set ϵ at 0.05 because they deemed this value ensured that with "precise knowledge of the database distribution and i.i.d. samples, the median raw recovery rate would be 100% except with probability 0.05." [28]. If ϵ is lowered, then all tokens would have the correct query in the list quicker with less leaked information, yet it would also increase the number of options and vice versa. As the attack already has significantly more options than the RPP-attack, the decision was made to not lower the ϵ. Increasing the ϵ was also not done, as then more tokens would not have the correct query in the list of options.

GLMP19 has a variable that is used in the arg_max method of their algorithm. The authors claim that the distribution for rank calculation converges to a Gaussian quickly [27]. We simulated this by using a constant that is checked against. The smaller the constant, the more accurate the algorithm, and the longer the run-time and vice versa. We set this value to $0.1 \cdot 10^{43}$ as we found that this gave good results without making the run-time unfeasible for most datasets. When we increased this value, we saw a decrease in document recovery performance. Decreasing it past this point does help, but also increases the run-time significantly. Given that it is already one of the longer-running algorithms, we decided to not do this.

6.5.6. Refine Speed

The variable in our refined aspect that impacts performance is *refine speed*. Throughout the experiments, this value was set to 1 which is the lowest value possible. This results in the most accurate results but also the longest run-time. 1 was chosen to give our algorithm the best chance of getting good recovery rates. If the refine speed is increased, the performance drops and the run-time goes down [18]. An attacker would thus need to see what aspect they deem most important. The run-time or the accuracy.

6.6. Run-time

The run-time results show that the RPP-attacks scale poorly with the increase in database and domain size. A larger domain impacts the attack as the increase in observed tokens is not linear. As the time to create the PQ-tree also rose with a larger document set, we can claim that the PQ-tree also scales poorly with an increase in the set of documents it has to handle. A significant portion of the base attack its time comes from the forming of the PQ-tree. Yet for the refined variant, the larger time aspect comes from our refining step. This shows that there is an improvement to be had in both the forming of the PQ-Tree as well as the refining step. A part of the refining step can be fixed by increasing the $refinespeed$. This means it does fewer iterations. Yet it will also mean that it refines less well [18].

The run-time results of the GLMP19 attack show that it takes about the same time for the PQ-Tree to be built for this attack as it did for the RPP=attacks. The run-time to get the actual values is significantly higher for GLMP19 compared to our basic attack. Yet it is significantly faster than the refined attack on the Enron and Lucene datasets. GLMP19 is not faster than both our attacks on the UCI dataset as GLMP19 did not finish within an acceptable time frame on that dataset. The attack, in its rank estimation, uses the total number of documents to deal with combinations in its calculation. There are an exponential number of calculations to be done relative to the number of documents. Given that the document set for UCI is very large, this is problematic. This shows that this attack might be unfeasible to run in its current form on datasets that have many documents.

Attacks such as KPT20 and GLMP18 have better run-time performances on average. For GLMP18 this will always be the case as this attack scales linearly to the number of possible queries and the number of observed tokens. This can be seen in table 5.3. There one can see that it runs fastest on the UCI dataset and that the run-time of Enron and Lucene is about equal.

KPT20 will not always have a better run-time. If the leakage of the scheme is $L2$, then KPT20 would be faster, because then the documents with the same reverse index key can be removed. The attack then finishes in 5.8 seconds while our basic attack finishes in 12.71 seconds. Our attack would also

work faster if we remove duplicate values at the start of the algorithm and put them back at the end. KPT20 can be modified to allow for duplicate document values. Because of the modifications needed, the mathematical problem becomes more complex and unfeasible to run on certain datasets [33]. We ran into the same problem where it did not finish on the UCI data within an acceptable time frame. The increase in time is visible in table 5.5, where with duplicates removed it takes 5.8 seconds and with the duplicates in the set, it takes 171.89 seconds. This is almost 30 times as slow. This effect would be even more on the UCI dataset due to how the problem needs to be solved. This shows that if the leakage is $L1$ that our attack is better to use than KPT20.

7

Limitations

7.1. Run-Time

As mentioned in the discussion, the run-time of the RPP-attacks can become quite large. This limits the capabilities of our attack. The KPT20 attack runs significantly faster under certain circumstances and is thus better suited if an attacker has less time available. Yet if the duplicate documents cannot be filtered out, then KPT20 is significantly slower than our attacks. Having our attack run faster will allow an attacker to run it on larger datasets and get the answer faster. One option to improve upon the performance would be to add multi-threading to the algorithm.

7.2. Countermeasures

As discussed, countermeasures negate parts, if not all of the RPP-Attack capabilities. It is thus important that the RPP-Attacks will be expanded to deal with these countermeasures. On the other hand, it clearly shows the need for countermeasures on servers to negate the attacker's capabilities.

7.3. Access Pattern

The RPP-attacks used the access pattern to analyze the range queries and the documents stored on the servers. Although our results are good, the access pattern is required to get the order of the documents. The attack in its current form would not succeed if we only saw the response volumes. Thus if a CSP uses a scheme that hides the access pattern and only reveals the response volume pattern, an attacker, would learn nothing regarding what it observes using the RPP-attack.

We are not the only attack that suffers from this, almost all of the attacks used in the comparison would fail if the access pattern is hidden. This shows that it is important that new attacks have to be developed that work with an access pattern hiding scheme. Kornaropoulos et al. have developed a new attack based on their old one [37] that can handle a response-hiding scheme [36]. For this attack, they use the same building blocks as in their earlier attack such as the estimators. The new attack looks at unique response volumes instead of access patterns.

Although it was not within our scope for this thesis to see how our attack could be extended to a response-hiding scheme, it is something we have looked at and thought about. A possible solution would be to get the elementary volumes as described in [28]. From these response volumes and known queries to response volume pairs we can, similarly, make estimations as to which query matches with what volume. Furthermore, it can also allow for document recovery.

7.4. Needed Knowledge

Although the RPP-attack has a good performance, and even better with the auxiliary information, it does require information upfront. The attack requires 2 known token-query pairs to start. Furthermore, the rank and combined auxiliary attacks can use a partial dataset. As stated earlier, this dataset might be somewhat unrealistic to be obtained. It is thus imperative that a similar dataset is considered. As could be seen in the results, the similar dataset scores well on the Lucene dataset, but worse on the Enron dataset while the KS for both is similar. The usage of the partial dataset results in similar results for both. It shows that for the auxiliary attacks to work, a good similar dataset needs to be available and that having a low KS similarity score does not guarantee success.

7.5. Similarity

The similar rank attack has significantly different results between the Enron, Apache and UCI data sets. It scores worst on the Enron dataset. This is due to how identical the similar dataset is. The similarity of documents does help a lot with the attack performance. Meaning that an attacker needs to be sure that the similar dataset is of high enough quality.

7.6. Attack Dependent on Query Distribution Used

Looking at the query distribution results in chapter 5, it is visible that query distribution plays a large role in how well the attack scores on the different metrics. The value-centred distribution scores the worst. These results show us that our attack works well given certain circumstances.

8

Future work

8.1. Two-Dimensional Range Queries

The RPP-attack currently works on single-dimension range queries. The current attack can be extended to work on multiple dimensions. The challenge here is that this needs to be done efficiently because otherwise, it can cause problems. This is a problem Dijkslag et al. also ran into when they extended single-keyword search attacks into a conjunctive setting. They showed that extending attacks to multiple dimensions can harm the accuracy [19].

If the conjunctive queries were handled by the server dealing with the two or more columns separately, and only at the end choosing the documents that were matched by all queries on the different columns, then the RPP-attack could be easily extended. The attack could be performed separately on each column. The value reconstruction attack would stay the same, the query recovery would change a tiny bit. There the attack would need to see which list of options was returned for the token on each column and only choose the ones that they all had in common. If the different columns are not processed separately, the above method and the RPP-attack in its current form will not work, and further research is required.

8.2. Volume Pattern

Our attack uses PQ-Trees and the access pattern for the attack to work. If the CSP would implement a scheme that hides the access pattern and only returns the volume pattern then the RPP-attack would not be able to discover any information. Schemes that do not return the access pattern will always leak the number of documents returned due to how SSE schemes work. Research can be done to extend the current attack to access pattern-hiding schemes. A possible solution would be to use the clique-finding technique described by Grubbs et. al. [28]. Using their techniques the different buckets could be obtained, but now with volumes instead of access patterns. Parts of the RPP-attack would also need to be altered as it is based on access patterns and not volume patterns. Yet this is possible. To use known query-token pairs, one would need to look for unique volumes etc. Yet, to further explore this was out of our scope.

8.3. Developing and Analyzing Countermeasures

In this thesis report, several countermeasures and their effectiveness were analyzed. Yet there are more countermeasures for range query schemes and normal single keyword schemes. Further analysis must be done on more countermeasures. This needs to be done to see how these countermeasures perform and what trade-offs there are from a computational and consumer ease standpoint. Such a thorough analysis would make it easier for CSPs and consumers to choose what kind of countermeasures they would like to be put in programs.

8.4. Run-Time Improvement

Even though the algorithm has been significantly improved in run-time since the first variant was made, there is still room for further improvements. Currently, the algorithm scales both in the number of documents and in the domain size. Where the Lucene and Enron data sets were run in 40 and 109 seconds respectively, it took almost 10 minutes for the refined RPP-attack to come to a result for the UCI Machine Learning dataset. Here the domain was smaller than with Enron and Lucene, but the database was significantly larger. In our tests, we noticed that increasing the domain by 10 resulted in significantly more queries being observed. This steep increase is also visible in Fig. 8.2 and 8.1 which is a zoomed-in version of Fig. 8.2. One can see here that the increase follows a similar shape as $O(N^2)$ while being lower. Yet it is significantly higher than $O(Nlog(N)$ which would be the next better complexity. Both aspects should be improved on in future work.

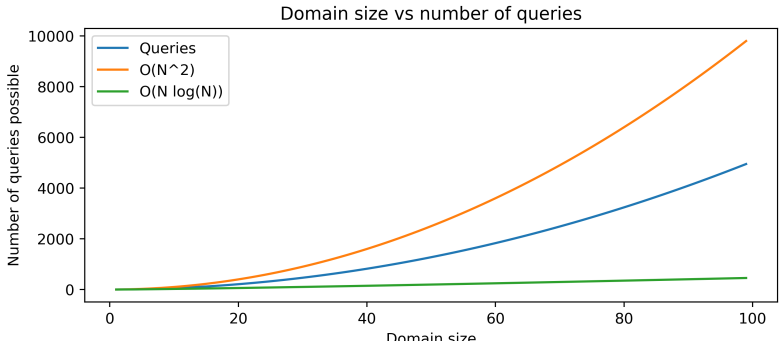


Figure 8.1: Increase in number of queries for a domain of 100

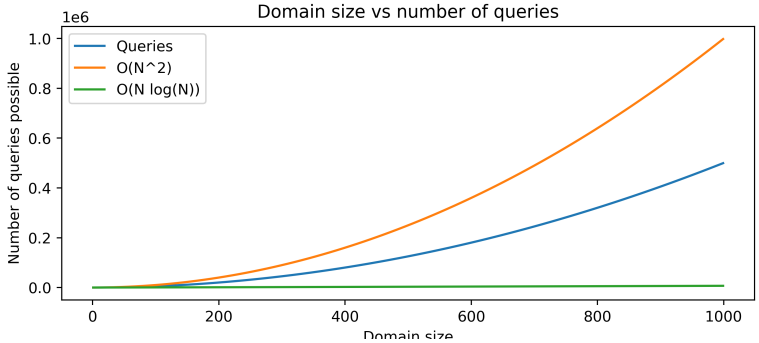


Figure 8.2: Increase in number of queries for a domain of 1000

8.5. Active Attack

Throughout the project, an active attacker was not explored. Looking at an active attacker would make the attacker have stronger assumptions but it would also increase their performance and accuracy. They would be able to learn more. For example, one could have an active attacker that can inject documents with specific range values and see in which bucket they show up. They can then assign values to these buckets. Another option would be to lower the possible values for the different buckets by injecting a document which has a value that is halfway in the domain. We have not explored this avenue due to looking at a static database, but we believe that this can increase the accuracy significantly. Especially in more sparse databases where the width of possible values per bucket can be quite large.

9

Conclusion

Throughout this thesis, we have developed two new attacks that use known token-query pairs as their base knowledge, after finding out what the current research has looked into. It was shown that these attacks can be easily extended using auxiliary information. This increases the attack performance even more. Furthermore, we have compared these attacks to current state-of-the-art attacks and shown that we outperform them. Lastly, the effect and importance of countermeasures were analyzed.

It is now possible to answer our sub-questions after which the main question can be answered: "**How can the leakage and structure of range searchable symmetric encryption schemes be exploited and what countermeasures work?**"

What are the leakage information of searchable symmetric encryption schemes supporting range queries? The leakage of SSE schemes that support range queries is the *access pattern*, *response volume*, and the *search pattern*. Both the access pattern and response volume have been exploited by attacks so far. Search pattern has not been looked into. Furthermore, the domain that gets leaked allows us to learn a lot about the database. We can already assign possible ranges to documents, as we know how many groups we have and how large the domain is. Further informatin can be found in chapter 2.

How can current attacks on range and single keyword schemes be optimized? While looking at current range query attacks in chapter 3, we observed that there are no attacks that actively exploit the query pattern leakage, while this has been exploited in normal keyword-based attacks. This was not a worthwhile avenue to explore as it requires very specific auxiliary information, which was deemed inapplicable in our case. Secondly, most attacks focus on access pattern, while a smaller amount focuses on response volume. Furthermore, a lot of attacks either look at similar or partial documents. None of the attacks uses known token-query pairs in their attack. Additionally, none allowed for more than one type of auxiliary information. Current attacks (on single keyword and range schemes) were optimized by similarly using known token-query pairs to Damie et al. [18]. This allowed us to create a new attack and introduce a new type of known data into the attack options for range query schemes in chapter 4. Lastly, the RPP-attack was made to be easily extendable for all types of information that can be learned from auxiliary datasets.

In what context can auxiliary information be used? As we have seen in attacks on range SSE schemes and on keyword schemes, a variety of auxiliary information can be used. Most attacks that use auxiliary information use it to get the response volume of queries, or use the rank information that they can retrieve from this. In the range SSE field, no one has used the volume of documents as auxiliary information. This is something that has been done in single keyword schemes by, for example, Lambregts et al. [39]. As rank information and volume information both seemed to work in other attacks, both were implemented in ours. Using this information, we claim that having 10 percent of the document set significantly increases the performance of the auxiliary RPP-attacks. Thus if the context allows an attacker to have auxiliary information available, they should exploit it.

How does the distribution of queries play a role? Kornaropoulos et al. showed that query distribution plays a significant role in the performance of the attacks [37]. Through our experiments and analysis in chapters 5 and 6, we saw that the query distribution plays a large role in the attack performance. In our experiments, not singular query distribution was always the best. For our attack, the short-range distribution outperformed the others, while for GLMP18 it was the value-centred distribution. For all attacks, the uniform distribution was the middle second or better. This shows that the query distribution is something every researcher has to take into account when developing new attacks, as they could make or break the performance of the attack and give an unfair bias.

What countermeasures could be used to evaluate the new attacks? Countermeasures are an important aspect of attacks. They tell us under what kind of circumstances the attacks perform well. To evaluate the attacks, the two countermeasures described by Markatou and Tamassia [42] were tested in section 5.5. Using these two countermeasures, we discovered and explained in section 6.3 that the Wrap-Around queries countermeasure completely negates our attack, but comes at a significant computational and bandwidth overhead. The GLMP19 and GLMP18 attacks are negated too, while KPT20 scores worse. The blocked queries countermeasure results in having all attacks score worse. Additionally, the security parameter has a large influence on how much the countermeasure impacts the performance. Lastly, the document volume hiding countermeasure only blocks the uses of volume information in the auxiliary attack, not our basic and refined variants.

To now answer the main question, we can state that the setup and query leakage can be exploited by looking at the domain and access patterns to create a successful attack that outperforms all other state-of-the-art attacks. Furthermore, both the Blocked and Wrap-Around queries countermeasures work to partially or fully negate the attack performance of the existing and our newly developed attacks.

Even though the setup and query leakage can be exploited in schemes that leak the access pattern, it does limit the capabilities of the attack. To further analyze this leakage, the same research can be formulated and used on schemes that only leak the volume pattern. Secondly, while the RPP-attack outperforms the other attacks, its run-time is a drawback. Future research could be done to see how the exploited information could be used in an even more efficient manner, making the attack more useful and efficient.

References

[1] Apache Pony Mail. *java-user@lucene.apache.org*. 2022. url: https://lists.apache.org/list.html?java-user@lucene.apache.org (visited on 12/24/2020).

[2] Alexandros Bakas and Antonis Michalas. *Nowhere to Leak: A Multi-client Forward and Backward Private Symmetric Searchable Encryption Scheme*. Vol. 12840 LNCS. 826093. Springer International Publishing, 2021, pp. 84–95. isbn: 9783030812416. doi: 10.1007/978-3-030-81242-3_5. url: http://dx.doi.org/10.1007/978-3-030-81242-3_5.

[3] Alexandros Bakas and Antonis Michalas. *Nowhere to Leak: A Multi-client Forward and Backward Private Symmetric Searchable Encryption Scheme*. Vol. 12840 LNCS. 826093. Springer International Publishing, 2021, pp. 84–95. isbn: 9783030812416. doi: 10.1007/978-3-030-81242-3_5. url: http://dx.doi.org/10.1007/978-3-030-81242-3_5.

[4] Alexandros Bakas and Antonis Michalas. "Power Range: Forward Private Multi-Client Symmetric Searchable Encryption with Range Queries Support". In: *Proceedings - IEEE Symposium on Computers and Communications* 2020-July (2020). issn: 15301346. doi: 10.1109/ISCC50000.2020.9219739.

[5] Laura Blackstone, Seny Kamara, and Tarik Moataz. "Revisiting Leakage Abuse Attacks". In: (2020). doi: 10.14722/ndss.2020.23103.

[6] Kellogg S. Booth and George S. Lueker. "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms". In: *Journal of Computer and System Sciences* 13.3 (1976), pp. 335–379. issn: 10902724. doi: 10.1016/S0022-0000(76)80045-1.

[7] Christoph Bösch et al. "A survey of provably secure searchable encryption". In: *ACM Computing Surveys* 47.2 (2014). issn: 15577341. doi: 10.1145/2636328.

[8] Christoph Bösch et al. "Conjunctive wildcard search over encrypted data". In: *Workshop on Secure Data Management*. Springer. 2011, pp. 114–127.

[9] Raphael Bost. "$\Sigma o \phi o \zeta$ - Forward secure searchable encryption". In: *Proceedings of the ACM Conference on Computer and Communications Security* 24-28-Octo (2016), pp. 1143–1154. issn: 15437221. doi: 10.1145/2976749.2978303.

[10] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. "Forward and backward private searchable encryption from constrained cryptographic primitives". In: *Proceedings of the ACM Conference on Computer and Communications Security* (2017), pp. 1465–1482. issn: 15437221. doi: 10.1145/3133956.3133980.

[11] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. "Forward and backward private searchable encryption from constrained cryptographic primitives". In: *Proceedings of the ACM Conference on Computer and Communications Security* (2017), pp. 1465–1482. issn: 15437221. doi: 10.1145/3133956.3133980.

[12] David Cash et al. "Highly-scalable searchable symmetric encryption with support for boolean queries". In: *Annual cryptology conference*. Springer. 2013, pp. 353–373.

[13] David Cash et al. "Leakage-abuse attacks against searchable encryption". In: *Proceedings of the ACM Conference on Computer and Communications Security* 2015-Octob (2015), pp. 668–679. issn: 15437221. doi: 10.1145/2810103.2813700.

[14] Javad Ghareh Chamani et al. "New constructions for forward and backward private symmetric searchable encryption". In: *Proceedings of the ACM Conference on Computer and Communications Security* (2018), pp. 1038–1055. issn: 15437221. doi: 10.1145/3243734.3243833.

[15] Javad Ghareh Chamani et al. "New constructions for forward and backward private symmetric searchable encryption". In: *Proceedings of the ACM Conference on Computer and Communications Security* (2018), pp. 1038–1055. issn: 15437221. doi: 10.1145/3243734.3243833.

[16] Guoxing Chen et al. “Differentially Private Access Patterns for Searchable Symmetric Encryption”. In: *Proceedings - IEEE INFOCOM* 2018-April (2018), pp. 810–818. issn: 0743166X. doi: 10.1109/INFOCOM.2018.8486381.

[17] William W Cohen. *Enron Email Dataset*. url: https://www.cs.cmu.edu/~enron/.

[18] Marc Damie, Florian Hahn, and Andreas Peter. “A highly accurate query-recovery attack against searchable encryption using non-indexed documents”. In: *Proceedings of the 30th USENIX Security Symposium* (2021), pp. 143–160.

[19] Marco Dijkslag et al. “Passive Query-Recovery Attack Against Secure Conjunctive Keyword Search Schemes”. In: *Applied Cryptography and Network Security*. Ed. by Giuseppe Ateniese and Daniele Venturi. Cham: Springer International Publishing, 2022, pp. 126–146. isbn: 978-3-031-09234-3.

[20] Maozhen Ding et al. “An efficient public key encryption with conjunctive keyword search scheme based on pairings”. In: *2012 3rd IEEE international conference on network infrastructure and digital content*. IEEE. 2012, pp. 526–530.

[21] Benjamin Edwards, Steven Hofmeyr, and Stephanie Forrest. “Hype and heavy tails: A closer look at data breaches”. In: *Journal of Cybersecurity* 2.1 (2016), pp. 3–14.

[22] Francesca Falzon et al. “Full Database Reconstruction in Two Dimensions”. In: *Proceedings of the ACM Conference on Computer and Communications Security* (2020), pp. 443–460. issn: 15437221. doi: 10.1145/3372297.3417275.

[23] Benjamin Fuller et al. “SoK: Cryptographically Protected Database Search”. In: *Proceedings - IEEE Symposium on Security and Privacy* 1414119 (2017), pp. 172–191. issn: 10816011. doi: 10.1109/SP.2017.10. arXiv: 1703.02014.

[24] Oded Goldreich and Rafail Ostrovsky. “Software protection and simulation on oblivious RAMs”. In: *Journal of the ACM (JACM)* 43.3 (1996), pp. 431–473.

[25] R. Groot Roessink. *Experimental review of the IKK query recovery attack : Assumptions, recovery rate and improvements*. July 2020. url: http://essay.utwente.nl/82324/.

[26] Ruben Groot Roessink, Andreas Peter, and Florian Hahn. *Experimental Review of the IKK Query Recovery Attack: Assumptions, Recovery Rate and Improvements*. Vol. 12727 LNCS. Springer International Publishing, 2021, pp. 155–183. isbn: 9783030783747. doi: 10.1007/978-3-030-78375-4_7. url: http://dx.doi.org/10.1007/978-3-030-78375-4_7.

[27] Paul Grubbs et al. “Learning to reconstruct: Statistical learning theory and encrypted database attacks”. In: *Proceedings - IEEE Symposium on Security and Privacy* 2019-May (2019), pp. 1067–1083. issn: 10816011. doi: 10.1109/SP.2019.00030.

[28] Paul Grubbs et al. “Pump up the volume: Practical database reconstruction from volume leakage on range queries”. In: *Proceedings of the ACM Conference on Computer and Communications Security* L (2018), pp. 315–331. issn: 15437221. doi: 10.1145/3243734.3243864.

[29] Agency for Healthcare Research and Quality.

[30] Chengyu Hu et al. “Forward secure conjunctive-keyword searchable encryption”. In: *IEEE Access* 7 (2019), pp. 35035–35048.

[31] Yong Ho Hwang and Pil Joong Lee. “Public key encryption with conjunctive keyword search and its extension to a multi-user system”. In: *International conference on pairing-based cryptography*. Springer. 2007, pp. 2–22.

[32] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. “Access pattern disclosure on searchable encryption: Ramification, attack and mitigation”. In: *Ndss* 20 (2012), pp. 1–15.

[33] Seny Kamara et al. “Cryptanalysis of Encrypted Search with LEAKER – A framework for LEakage AttacK Evaluation on Real-world data”. In: November (2019), pp. 1–34.

[34] Georgios Kellaris et al. “Generic attacks on secure outsourced databases”. In: *Proceedings of the ACM Conference on Computer and Communications Security* 24-28-Octo (2016), pp. 1329–1340. issn: 15437221. doi: 10.1145/2976749.2978386.

[35] Florian Kerschbaum and Anselme Tueno. *An Efficiently Searchable Encrypted Data Structure for Range Queries*. Vol. 11736 LNCS. Springer International Publishing, 2019, pp. 344–364. isbn: 9783030299613. doi: 10.1007/978-3-030-29962-0_17. arXiv: 1709.09314. url: http://dx.doi.org/10.1007/978-3-030-29962-0_17.

[36] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. "Response-hiding encrypted ranges: revisiting security via parametrized leakage-abuse attacks". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 1502–1519.

[37] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. "The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution". In: *Proceedings - IEEE Symposium on Security and Privacy* 2020-May (2020), pp. 1223–1240. issn: 10816011. doi: 10.1109/SP40000.2020.00029.

[38] Marie Sarah Lacharite, Brice Minaud, and Kenneth G. Paterson. "Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage". In: *Proceedings - IEEE Symposium on Security and Privacy* 2018-May (2018), pp. 297–314. issn: 10816011. doi: 10.1109/SP.2018.00002.

[39] Steven Lambregts et al. "VAL: Volume and Access Pattern Leakage-Abuse Attack with Leaked Documents". In: *European Symposium on Research in Computer Security*. Springer. 2022, pp. 653–676.

[40] Yuxi Li et al. "Integrity-verifiable conjunctive keyword searchable encryption in cloud storage". In: *International Journal of Information Security* 17.5 (2018), pp. 549–568.

[41] Chang Liu et al. "Search pattern leakage in searchable encryption: Attacks and new construction". In: *Information Sciences* 265 (2014), pp. 176–188. issn: 00200255. doi: 10.1016/j.ins.2013.11.021. url: http://dx.doi.org/10.1016/j.ins.2013.11.021.

[42] Evangelia Anna Markatou and Roberto Tamassia. "Mitigation Techniques for Attacks on 1-Dimensional Databases that Support Range Queries". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11723 LNCS (2019), pp. 231–251. issn: 16113349. doi: 10.1007/978-3-030-30215-3_12.

[43] Evangelia Anna Markatou et al. *Reconstructing with Less: Leakage Abuse Attacks in Two Dimensions*. Vol. 1. 1. Association for Computing Machinery, 2021, pp. 2243–2261. isbn: 9781450384544. doi: 10.1145/3460120.3484552.

[44] Simon Oya and Florian Kerschbaum. "Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption". In: *Proceedings of the 30th USENIX Security Symposium* (2021), pp. 127–142. arXiv: 2010.03465.

[45] Yu Peng et al. "Dynamic Searchable Symmetric Encryption with Forward and Backward Privacy". In: *Proceedings - 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2021* (2021), pp. 420–427. doi: 10.1109/TrustCom53373.2021.00070.

[46] Jay G Ronquillo et al. "Health IT, hacking, and cybersecurity: national trends in data breaches of protected health information". In: *JAMIA open* 1.1 (2018), pp. 15–19.

[47] Saeed Sedghi et al. "Searching keywords with wildcards on encrypted data". In: *International Conference on Security and Cryptography for Networks*. Springer. 2010, pp. 138–153.

[48] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. "Practical techniques for searches on encrypted data". In: *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*. IEEE. 2000, pp. 44–55.

[49] Emil Stefanov et al. "Path ORAM: an extremely simple oblivious RAM protocol". In: *Journal of the ACM (JACM)* 65.4 (2018), pp. 1–26.

[50] Shi Feng Sun et al. "Practical backward-secure searchable encryption from symmetric puncturable encryption". In: *Proceedings of the ACM Conference on Computer and Communications Security* (2018), pp. 763–780. issn: 15437221. doi: 10.1145/3243734.3243782.

[51] UCI. *Bank Marketing Data Set*. 2007. url: http://archive.ics.uci.edu/ml/datasets/Bank+Marketing (visited on 08/20/2022).

[52] UCI. *UCI Machine Learning Repository*. 2007. url: http://archive.ics.uci.edu/ml/index.php (visited on 08/20/2022).

[53] Xiangyu Wang et al. “Search Me in the Dark: Privacy-preserving Boolean Range Query over Encrypted Spatial Data”. In: *Proceedings - IEEE INFOCOM* 2020-July (2020), pp. 2253–2262. issn: 0743166X. doi: 10.1109/INFOCOM41043.2020.9155505.

[54] Yang Yang et al. “Flexible wildcard searchable encryption system”. In: *IEEE Transactions on Services Computing* 13.3 (2017), pp. 464–477.

[55] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. “All your queries are belong to us: The power of file-injection attacks on searchable encryption”. In: *Proceedings of the 25th USENIX Security Symposium* (2016), pp. 707–720.

A

Attack Diagram

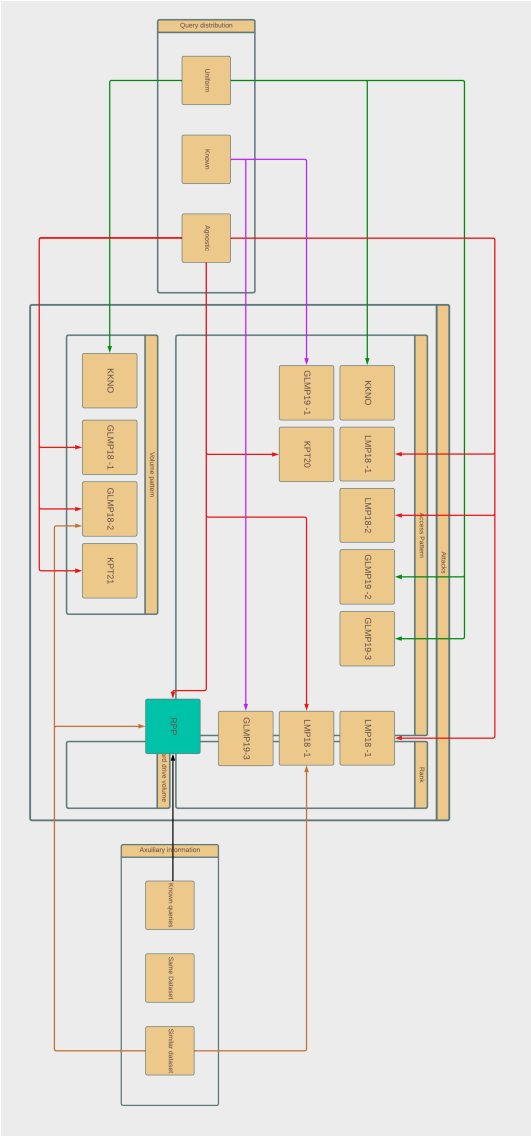


Figure A.1: Attack diagram

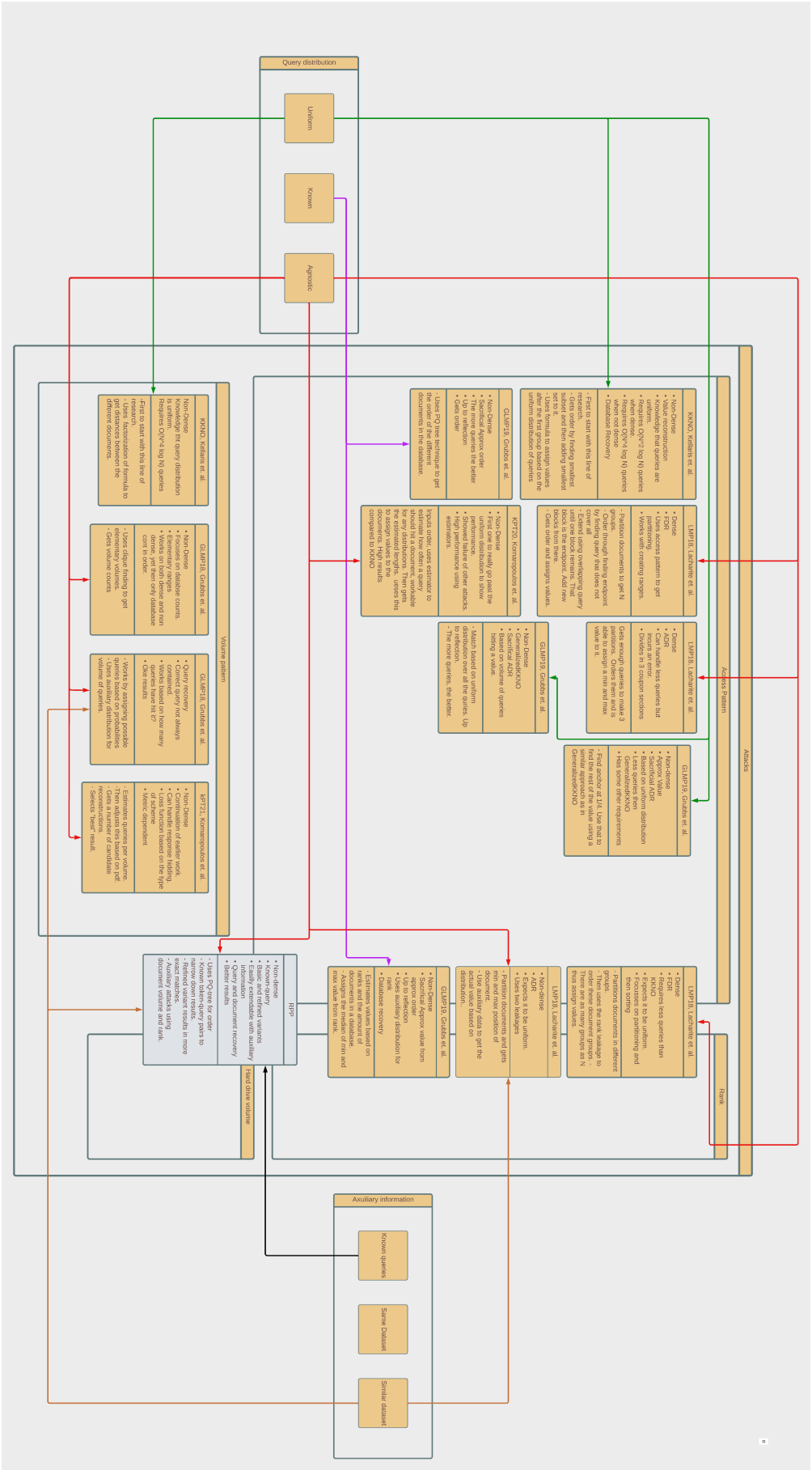
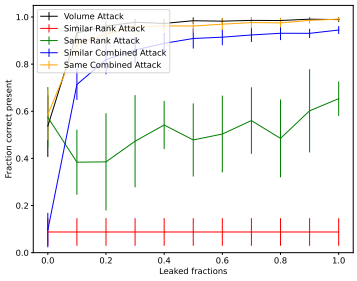


Figure A.2: Attack diagram

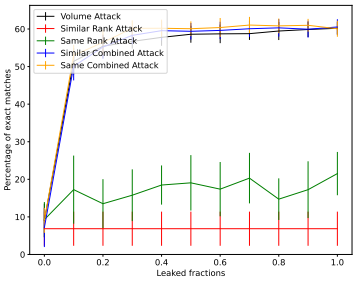
B

Figures

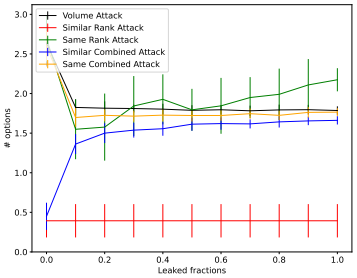
B.1. Enron



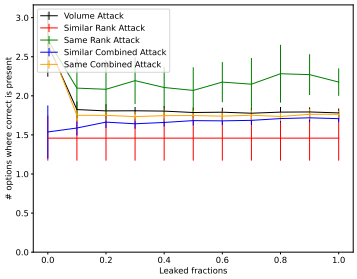
(a) Fraction of correct query present in list (higher is better)



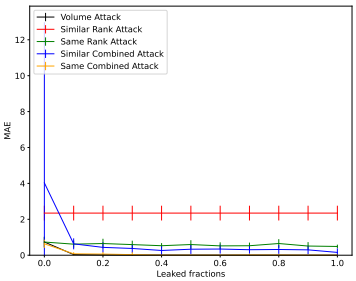
(b) Percentage of exact matches per densities (higher is better)



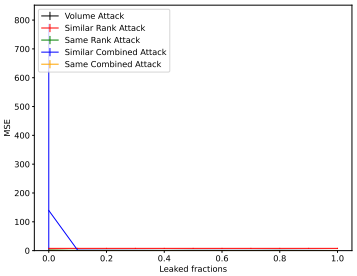
(c) Number of options per token



(d) Number of options where the correct query is present (higher is better)



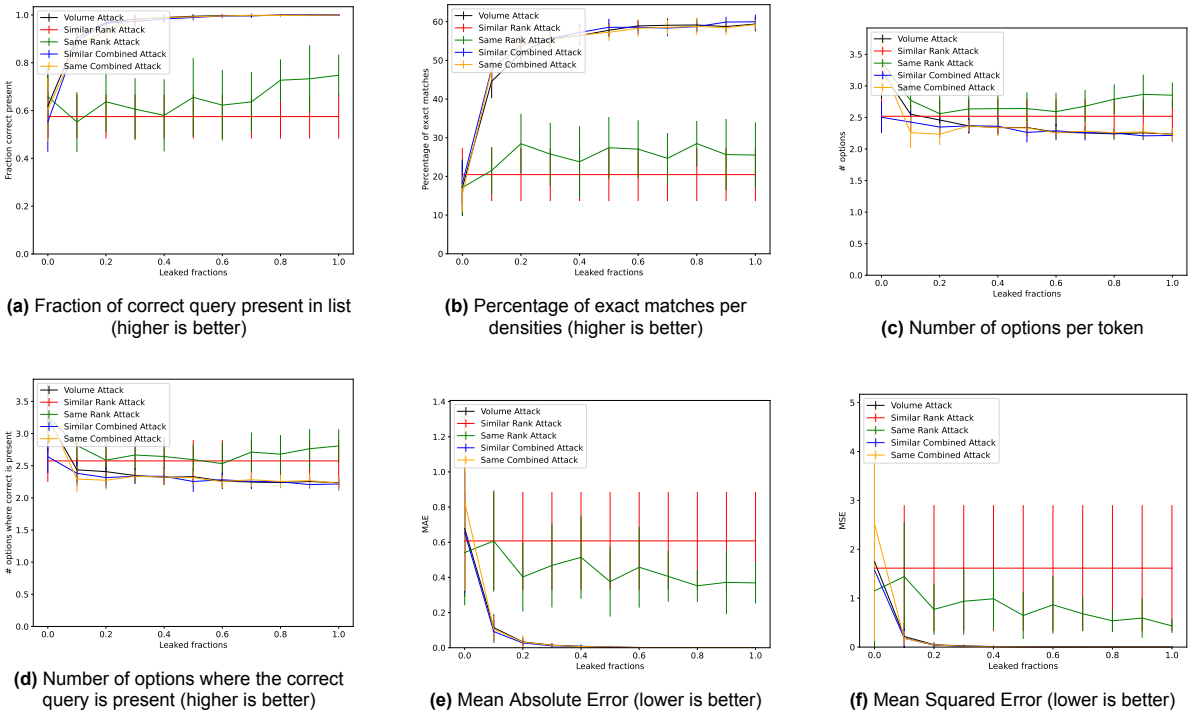
(e) Mean Absolute Error (lower is better)



(f) Mean Squared Error (lower is better)

Figure B.1: Comparisons of metrics for the different auxiliary attacks with the refined variant

B.2. Lucene



(a) Fraction of correct query present in list (higher is better)

(b) Percentage of exact matches per densities (higher is better)

(c) Number of options per token

(d) Number of options where the correct query is present (higher is better)

(e) Mean Absolute Error (lower is better)

(f) Mean Squared Error (lower is better)

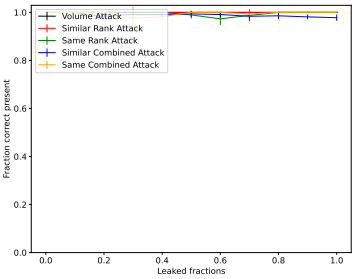
Figure B.2: Comparisons of metrics for the different auxiliary attacks with the refined variant

B.3. UCI

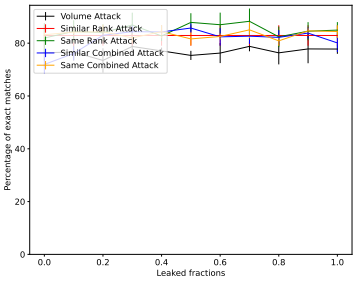
(a) MAE

(b) MSE

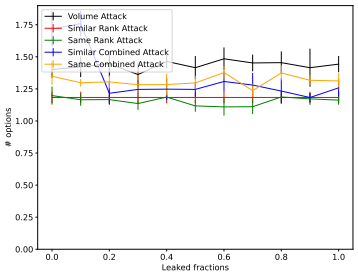
Figure B.3: UCI comparison of own attacks vs KPT20



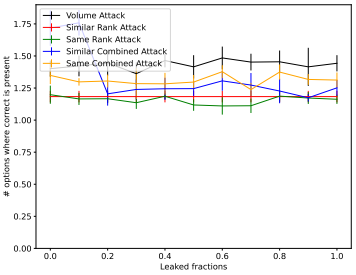
(a) Fraction of correct query present in the list (higher is better)



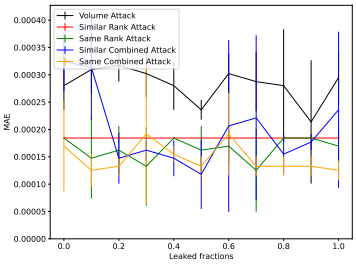
(b) Percentage of exact matches per densities (higher is better)



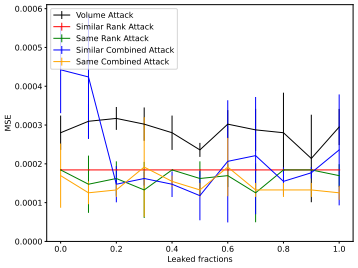
(c) Number of options per token



(d) Number of options where the correct query is present (lower is better)

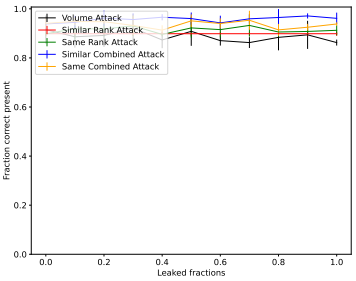


(e) Mean Absolute Error (lower is better)

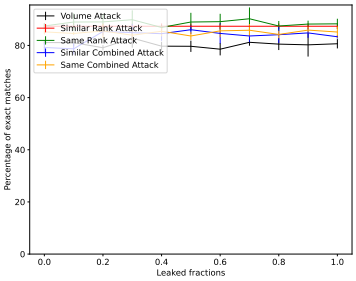


(f) Mean Squared Error (lower is better)

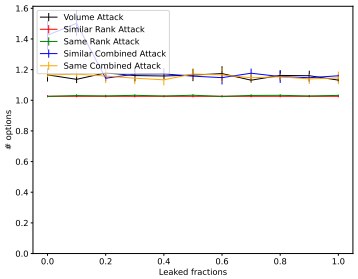
Figure B.4: Comparisons of metrics for the different auxiliary attacks with the basic variant



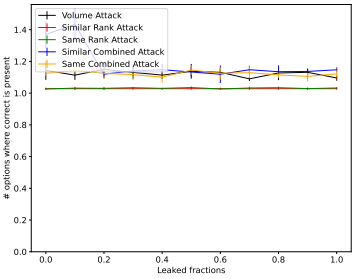
(a) Fraction of correct query present in list (higher is better)



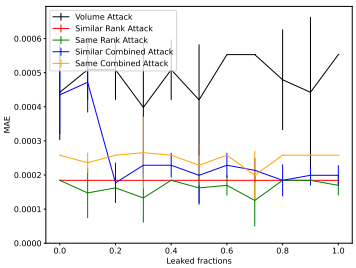
(b) Percentage of exact matches per densities (higher is better)



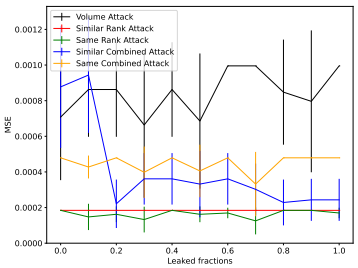
(c) Number of options per token



(d) Number of options where the correct query is present (higher is better)



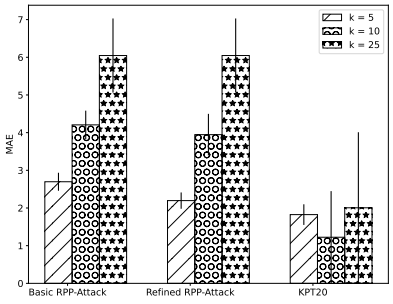
(e) Mean Absolute Error (lower is better)



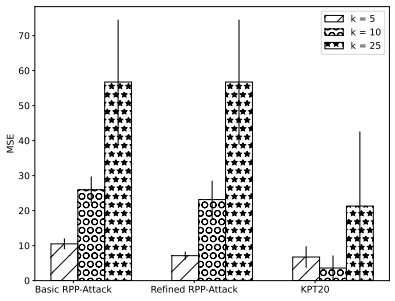
(f) Mean Squared Error (lower is better)

Figure B.5: Comparisons of metrics for the different auxiliary attacks with the refined variant

B.4. Countermeasures



(a) Mean Absolute Error (lower is better)



(b) Mean Squared Error (lower is better)

Figure B.6: Comparisons of metrics for different values of k for own attack compared to KPT20 on UCI