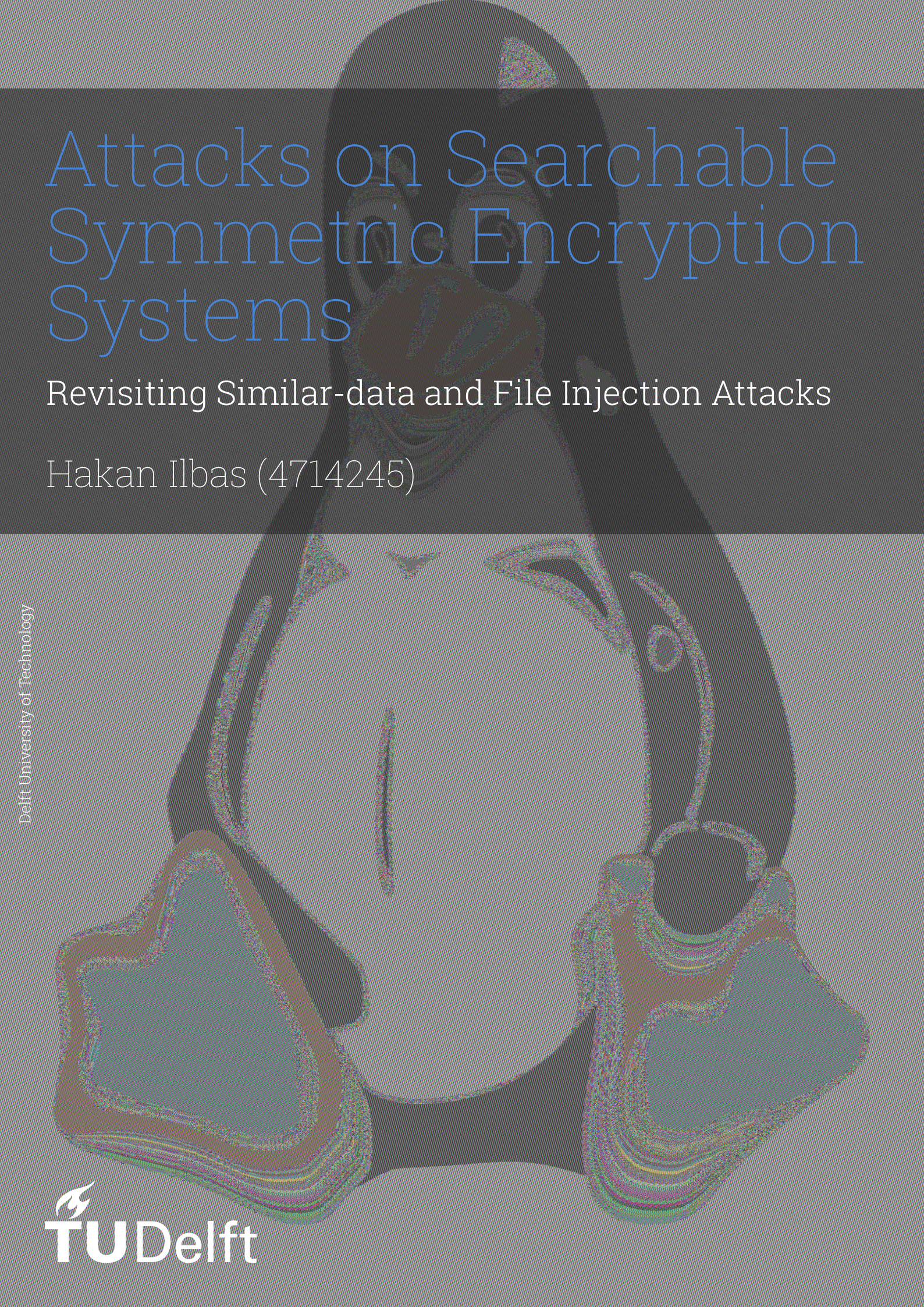
Attacks on Searchable Symmetric Encryption Systems

Revisiting Similar-data and File Injection Attacks

Hakan Ilbas (4714245)



Delft University of Technology

TU Delft

Attacks on Searchable Symmetric Encryption Systems

Revisiting Similar-data and File Injection Attacks

by

Hakan Ilbas (4714245)

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday December 7, 2022 at 01:00 PM.

Student number:	4714245	
Project duration:	December, 2021 – December, 2022	
Thesis committee:	Prof. dr. ir. G. Smaragdakis,	TU Delft, thesis advisor
	Dr. K. Liang,	TU Delft, daily supervisor
	Dr. J. Decouchant,	TU Delft

Cover:	The Encrypted Pinguin by Filippo Valsorda under CC BY-SA 4.0 (Modified)
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

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

TU Delft

Preface

Dear reader, I am writing this preface in the last stage of my thesis. Last year has been quite challenging. From knowing not even what Searchable encryption is about, to knowing almost every single detail about it. I spent a lot of time reading tons of papers and trying to understand how the attacks work. It has been quite a journey in which I kept pushing myself to work harder and harder, while trying not to lose my social life as well. But now it is time for my work to shine with this thesis, which marks the end of my academic career.

I would like to thank professor Kaitai Liang, who has introduced me to the concept of Searchable Encryption and who was my daily supervisor. Furthermore I would like to thank Huanhuan Chen, who was my daily co-supervisor, who helped me to understand some concepts and shined light onto topics from a different perspective. I would also like to thank professor Georgios Smaragdakis for becoming my thesis advisor and professor Jérémie Decouchant for becoming part of my thesis committee.
Finally, I would like to thank my friends and family for their continued support for me in the last year.

Hakan Ilbas (4714245)
Delft, December 2022

Contents

1

Abstract

The amount of data individuals create keeps increasing every year to the point that the data cannot be stored on a single device anymore. Cloud storage provides a solution for this problem, but not everybody wants the cloud storage service providers to peek at their data and they thus encrypt their data before storing it on the service provider's servers. Unfortunately, due to the way encryption works, the users are not able to perform simple actions on their data, like for example keyword search. However with Searchable Symmetric Encryption (SSE) the users can still perform keyword search on their data when their data is encrypted. With the use of SSE, there is some information that is being exposed about the data that is being stored on the system, called leakage. This leakage can be used by attackers in an attack to perform query recovery.

Currently existing attacks are mostly known-data attacks which assume that the attacker already has access to a large part of the plaintexts stored on the system. However this is very unlikely in real-world scenarios. A few papers focus on similar-data attacks which have a slightly different assumption. With similar-data attacks, the assumption is that the attacker has a similar document set to the document set stored on the SSE system. These attacks are therefore more realistic than known-data attacks, but the best similar-data attack still has some flaws.

Therefore, in this thesis, we propose a new attack that is based on an already existing similar-data query recovery attack. This new attack is a combination of a file injection attack and a similar-data attack. This new attack achieves a higher accuracy than the best similar-data and known-data attacks, while injecting only a few files into the SSE system. To the best of our knowledge this is the first similar-data attack with a file injection component. The new attack is also more resilient to countermeasures such as padding and obfuscation.

2

Introduction

Every day roughly 2.5 quintillion bytes of data are created [35]. This is a 2.5 with 18 zeros. With the amount of people on earth and the amount of data individuals create increasing, this number will only grow and grow in the upcoming years. This has a few technical problems. The first one being that not all of your data can be stored on a single device anymore. Cloud storage provides a solution for this problem. With cloud storage, users can cheaply store and access their data online, which is now stored in the cloud instead of locally being stored only on their devices. But, as users become more privacy aware, they do not want the cloud storage service providers to take a look at their data, so a logical step is to encrypt their data before uploading it to the cloud. But by encrypting their data, the users lose the ability to perform simple actions on their data, like for example keyword search. A solution for this are Searchable Symmetric Encryption (SSE) schemes. These schemes provide the user the ability to still perform keyword search on their encrypted data. However these schemes expose some information about the content that is stored on the SSE systems, and this may lead to attacks.

To understand how attacks on SSE systems work, one would first need to understand what SSE systems are and what they are used for. A short description is already given above in the form of an example, but imagine a more concrete scenario. A user has a lot of important files and the user needs to regularly search over these files (performing keyword search). These files can be e-mails or other important documents. Eventually the user reaches a point in which managing all of the data on his own becomes too expensive (high operation costs or too unsafe to store everything locally), and thus the user decides to store the important files on another server, hosted by a third party. However since these files contain confidential information, the user wants to encrypt the data before uploading it to the third party's servers.

The problem with encryption is that once the user encrypts the files, the ability to perform keyword search on those files becomes impossible, since encrypted files are unreadable without the key it is encrypted with. The only way for the user to perform keyword search again, would be to download and decrypt everything that is stored on the third party's server, but this defeats the purpose of storing the data on the third party's server and it also becomes very expensive (too time consuming while also being very resource intensive since all files need to be decrypted as well).

Searchable symmetric encryption, as the name suggests, provides a solution for this problem, as with SSE systems it is still possible to perform keyword search over encrypted files with some very clever techniques which will be explained in chapter 3.

But SSE systems are susceptible to leakage, in the form of access pattern leakage, search pattern leakage and other leakage patterns. There also exist different SSE systems which leak more information than others. This leakage can be used by an adversary to determine the keywords that the user of the system is querying, since when the user performs keyword search, he encrypts the keywords because he does not want the third party to know what is contained in the files. The attack the attacker performs to determine the keywords of the encrypted queries is called query recovery. Another cate-

gory of attacks are File Injection Attacks which do not use leakage, but rather depend on the adversary injecting files into the SSE system. These attacks are also used for query recovery.

SSE systems are very interesting since they provide a solution for a growing problem for a lot of companies which want to outsource their data, but still want the ability to search over it. However, because of the leakages using and storing data on SSE systems also comes with a risk.

This thesis will focus on creating a new attack on SSE systems. This new attack will have a higher accuracy on query recovery than currently existing attacks on SSE systems. The main attack will be based on a similar-data attack with a file injection component. More information about what similar-data attacks are and what file injection is, will be given in further chapters.

2.1. Relevance

SSE systems are already in use by a number of companies such as Ciphercloud [18] (now Lookout), BitGlass [31] etc. The companies provide the use of SSE systems as a service to their customers. This means that attacks on SSE systems are very relevant since these companies already have a lot of customers. If an attacker would be able to perform a successful attack, this would thus mean that he would have access to the search queries on the data of the customers of these companies. However the attacker could also be the service providers themselves. By getting to know which sorts of documents are stored on their systems, they could sell this information to digital advertisement companies which would be able to then target their ads better.

SSE systems are used by users who have a lot of data and who do not want to store it locally. Since we currently live in a day and age where the amount of created data keeps growing, and since it is too dangerous to keep only a single copy of that data locally (be it that the device might get lost or that it is infected with ransomware etc.), the chance of SSE systems being used will become larger and thus this also increases the relevance of attacks on SSE systems.

2.2. Stakeholders

For attacks on SSE systems, there are three main stakeholders, namely:

- The attackers, since if they would be able to steal the data, they could threaten the data owner or service provider by selling it or making it public. This could cause some major problems depending on what data was being stored. The attackers can do this to earn some quick money, or to demolish a company or entity, just for fun.
- The SSE service providers, since they provide the SSE system as a service to their customers. However if the systems they provide have major security flaws (such as being susceptible to query recovery attacks), it would lead to a loss of customers and potential damages due to lawsuits etc. Therefore they have a major stake in attacks on SSE systems, since everything needs to be done in order to prevent them. In the case of an attack, all customers could stop using the service to switch to one of their competitors and new clients would not sign up for the same reason.
- The customers (or data owners), since it is eventually their search queries on their data that will be stolen in the case of a successful attack. Therefore they have the largest stake in attacks on SSE systems. Depending on how many users SSE systems will have in the future, their stake will keep increasing and increasing if more users make use of SSE systems.

2.3. Contributions

This thesis provides an extensive comparison between the currently known SSE attacks to discover each attacks its pros and cons. Furthermore this thesis provides knowledge about the weaknesses of the refined score attack described in [9] and tries to improve its accuracy since it is one of the few similar-data attacks. The most interesting part of the thesis will be the new attack, which will be a similar-data attack with a file injection component. To the best of the authors knowledge this has not been done before at the time of writing and therefore it will be the first similar-data file injection attack. Furthermore the new attack will be evaluated on three different datasets and it will be compared to the known-data and similar-data attacks that currently have the highest query recovery accuracy.

The rest of the thesis is organized as follows. In the next chapter SSE attacks are explained in great detail. In chapter 4 the related work to this thesis is discussed. Chapter 5 gives the research question of this thesis and in chapter 6 the refined score attack and its weaknesses are explained in greater detail. In chapter 7 the new attack that is created will be explained and chapter 8 will describe the conducted experiments. After that chapter 9 will show the results of the new attack in comparison to already existing existing SSE attacks. These results will be used in chapter 10 to answer the research question.

3
Background

3.1. Searchable Symmetric Encryption

The idea of SSE systems has already been explained in the previous section. But in this section it will be explained in a more formal way with more details. Bear in mind that the attack model described in this work and other works, does not depend on the underlying searchable encryption scheme. The attack models succeed as long as the searchable encryption scheme reveals some patterns to the attacker. The same holds for the encryption schemes. The attack models do not consider the way the documents and queries (also called trapdoors) are encrypted. As far as the attacks are concerned the way everything is encrypted does not matter, since the attacks do not focus on cracking the encryption algorithms, but rather focus on leakages created by SSE systems when using them.

The reason why there is this thesis we focus on Searchable Symmetric Encryption and not on other forms of searchable encryption such as asymmetric encryption [1, 4, 5, 19, 39, 41], is because asymmetric encryption is slower in terms of speed than symmetric encryption. It takes a longer amount of time for the asymmetric encryption algorithms to decrypt the encrypted documents when they arrive at the user. Furthermore, for symmetric encryption shorter key lengths can be used which makes it easier to use for the users, and the algorithms are usually simpler than asymmetric encryption algorithms, which makes it easier for legacy systems with less computational resources to incorporate SSE schemes. Using Asymmetric encryption becomes cumbersome when frequently decrypting documents when you receive them from the third party. That is the reason for why SSE is much more popular and hence the reason why this thesis focuses on SSE attacks rather than asymmetric searchable encryption attacks. .

3.2. Simplified SSE attack model

In the attack model on SSE systems, we have Alice and Bob. Alice wishes to perform keyword search on her document set. Alice decides to use an SSE system and to do this, Alice first builds a modified inverted matrix over the set of her documents. This modified inverted index can be seen as a binary table, in which the rows are indexed by the keyword set and the columns are indexed by the document set (this can be the other way around as well). All entries in this table are 0, except if a keyword occurs in a document. Then this entry is 1. Formally, the $(i, j)^{th}$ entry of the inverted index is 1 iff the i^{th} keyword appears in the j^{th} document and 0 otherwise.

In this model, Bob, who is the server, performs keyword search based on this inverted index. Alice sends the inverted index to Bob by encrypting the rows of the matrix independently, since she does not want Bob to know what information is contained in her documents. Furthermore, she encrypts the keywords as well when she is sending a query to Bob, by applying a trapdoor function on them. When Alice wants to perform keyword search, she applies the trapdoor to the keyword and sends this to Bob. Bob then looks up the encrypted keyword in the inverted index and sends Alice the encrypted row of the inverted index. Alice decrypts this row and asks for the appropriate set of encrypted documents.

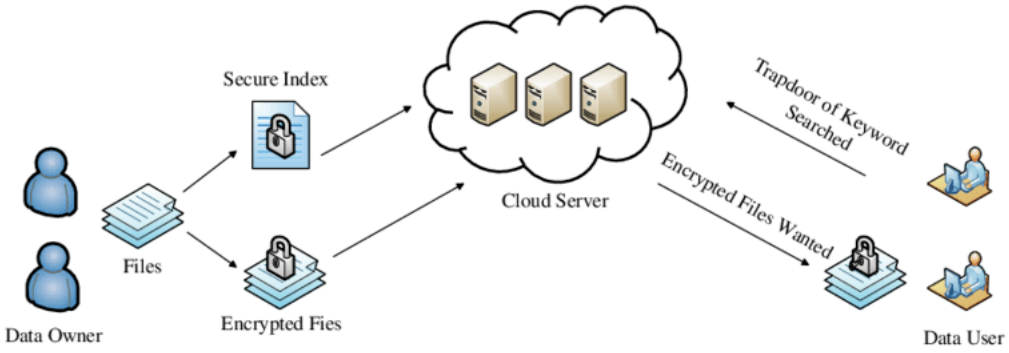


Figure 3.1: A typical SSE system [15].

Bob then sends the encrypted documents back to Alice.

The last two steps can also be merged into a single step, in which Bob sends the encrypted documents when he receives a query without Alice decrypting the row first. This does not make a huge difference to the way the inverted index is stored. This is how a regular SSE system works regularly and this is also how most of the literature on SSE attacks describe SSE systems. A visual representation is shown in figure 3.1. Here the data owner and data user are two separate entities, but they can also be one entity as is the case with Alice and Bob in the example given earlier. During this work, the term query recovery will be used. This refers to the attacker trying to predict the keyword that belongs to the (encrypted) query that the user is sending to the server.

3.3. Notation

Notation	Meaning
k	the security parameter
$\mathbb{D}$	the entire document collection
$\mathbb{W}$	keyword space
#S	cardinality of S
$2^{[n]}$	the power set
D	a document in the document collection
D	the document collection
w	keyword
Q	set of Queries
$\|D\|_w$	word length of a document
q_i	i^{th} element of q
f	amount of files to inject

3.4. Leakage

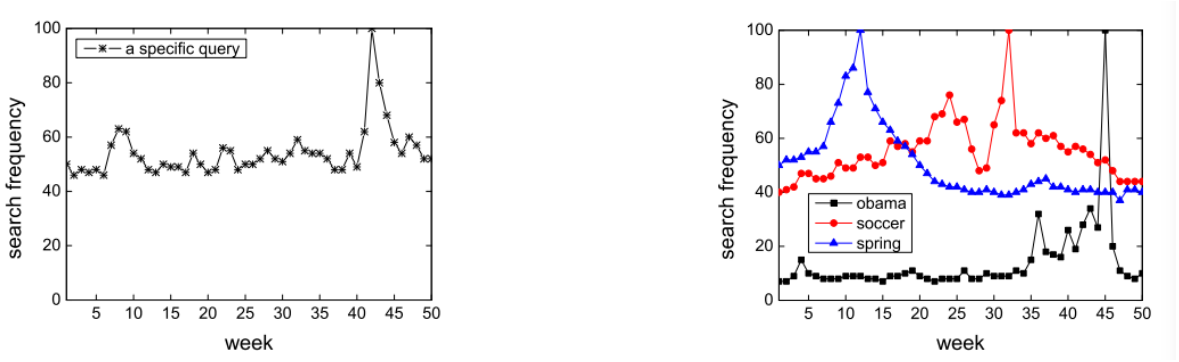
All SSE systems expose some information, called leakage. This leakage is about the plaintext. The leakage might be exposed to the SSE service provider or to some man in the middle. Leakage helps the attackers to perform an attack on the SSE systems. These leakages (or patterns) come in different forms.

3.4.1. Search pattern leakage

The search pattern can be seen as a frequency of how many times a certain query has been queried by the user in a certain time frame. This time frame can be different as long as it is the same time frame you compare it to with other queries. Concretely, the search pattern is described as the information about whether any two queries are generated from the same keyword or not [21]. In simpler words it represents the search frequency of a query in a specific time frame, as can be seen in 3.2a.

A basic frequency analysis attack can be performed with the search pattern, by comparing the search pattern of the user queries and the search pattern of another system that can be used to perform queries, such as Google. With Google Trends [12], search patterns can be exported and analyzed of every query performed on Google's search engine. By comparing these two search patterns it would theoretically be possible to make some predictions for the keywords of the encrypted queries. The idea is shown in 3.2. This is more or less the attack described in [21], but this SSE attack has a very bad accuracy. The attack performs frequency analysis on both search patterns, and the closest matching pairs (query on Google Trends and encrypted queries on the SSE system) will be the predictions made by the attack. Since the attack is also relatively simple, it will not be discussed in this paper anymore.

Formally the search pattern leakage can be described as the query equality pattern. Its formal definition is given in definition 1.



(a) Search frequency of an encrypted query.

(b) Search frequency of some queries.

Figure 3.2: Search frequency over time.

Definition 1. *The* query equality pattern *is the function family qeq =* $\{qeq_{k,t}\}_{k,t\in\mathbb{N}}$ *with* $qeq_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \rightarrow \{0,1\}^{t\times t}$ *such that* $qeq_{k,t}(\boldsymbol{D}, w_1, ..., w_t)$ = M *where* M *is a binary* $t \times t$ *matrix such that* M*[*i, j*] = 1 if* $w_i = w_j$ *and* M*[*i, j*] = 0 if* $w_i \neq w_j$*. The query equality pattern is referred to as the search pattern in the SSE literature [3].*

3.4.2. Access pattern leakage

The access pattern is the information about which encrypted queries occur in which encrypted documents for each of the user queries. Essentially, the access pattern can be seen as a mapping from user queries (which are encrypted), to encrypted documents. In almost all attacks in the literature, the access pattern leakage is used. In [28] a leakage hierarchy for the access pattern is created. This will be discussed in a later subsection. Formally the access pattern leakage can be described as the identifier pattern. Its formal definition is given in definition 2.

Definition 2. *The* identifier pattern *is the function family rid =* $\{rid_{k,t}\}_{k,t\in\mathbb{N}}$ *with* $rid_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \rightarrow [2^{[n]}]^t$ *such that* $rid_{k,t}(\boldsymbol{D}, w_1, ..., w_t)$ *= (ids(*w_1*), ..., ids(*w_t*)). The identity pattern is referred to as the access pattern in the SSE literature [3].*

In [11, 25], the authors present their oblivious RAM mode, in which no access patterns are revealed when the user performs keyword search over encrypted data. This SSE scheme is referred to as ORAM. The problem however is that there is a lot of overhead when using the ORAM protocol, making it a less preferred choice. Therefore ORAM schemes will not be discussed anymore in this work. Most research also does not focus on attacks on ORAM schemes.

3.4.3. Response length leakage

The response length leakage, can be seen as a part of the access pattern leakage, since the response length leakage, reveals for every query, the number of documents that contain it. The Count attack [28] uses the response length to create its base mappings, as will be explained later. If the response length is unique for some of the queries, it will be easier for the attacker to predict the correct keyword-trapdoor pairs. The formal definition of the response length leakage or response length pattern is given in definition 3.

Definition 3. *The* response length pattern *is the function family rlen =* $\{rlen_{k,t}\}_{k,t\in\mathbb{N}}$ *with* $rlen_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \to \mathbb{N}$ *such that* $rlen_{k,t}(\boldsymbol{D}, w_1, ..., w_t) = (\#\boldsymbol{D}(w_1), ..., \#\boldsymbol{D}(w_t))$ *[3].*

3.4.4. Volume pattern

Each document in the document set has a certain size, or volume. This number varies depending on how large a document is. The volume can for example be the amount of words or characters in the document, or the total size in bytes. Some documents may have a unique volume, and this might make it interesting in some attacks to create encrypted document and document mappings in known-data attacks. The volume pattern reveals, for each query, the volumes of the documents that contain it. As explained earlier, ORAM mitigates some attacks since attackers cannot identify which documents are returned as a response to a query. However ORAM cannot defeat the volume pattern, as attackers are still able to observe the volume in a response [29]. The volume pattern is not unhidable, since there are some techniques to hide it [16]. The volume pattern is an important pattern for some encryption schemes [22, 6, 7, 17]. The formal definition of the volume pattern is given in definition 4.

Definition 4. *The* volume pattern *is the function family vol =* $\{vol_{k,t}\}_{k,t\in\mathbb{N}}$ *with* $vol_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \to \mathbb{N}^t$ *such that* $vol_{k,t}(\boldsymbol{D}, w_1, ..., w_t) = ((|D|_w)_{D\in\boldsymbol{D}(w_1)}, ..., (|D|_w)_{D\in\boldsymbol{D}(w_t)})$ *[3].*

3.4.5. Total volume pattern

The total volume pattern is similar to the volume pattern. The total volume pattern reveals, for each query, the sum of the volumes of the documents that contain it. The formal definition of the total volume pattern is given in definition 5

Definition 5. *The* total volume pattern *is the function family vol =* $\{tvol_{k,t}\}_{k,t\in\mathbb{N}}$ *with* $tvol_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \to \mathbb{N}^t$ *such that* $tvol_{k,t}(\boldsymbol{D}, w_1, ..., w_t) = (\sum_{D\in\boldsymbol{D}(w_1)} |D|_w, ..., \sum_{D\in\boldsymbol{D}(w_t)} |D|_w)$ *[3].*

3.5. Leakage Hierarchy

Different sorts of SSE systems have been created. Some leak more information than others. Therefore a leakage hierarchy for SSE systems is created by [28] to categorize the SSE systems. This leakage hierarchy defines a set of leakage profiles, ranging from L4 to L1. The weakest leakage profile (and thus the profile that leaks the most information) is leakage profile L4. The strongest (and thus the profile that leaks the least amount of information) is leakage profile L1. A visual representation of how the L4, L2 and L1 schemes work, can be found in figure 3.3.

3.5.1. L4 Leakage Profile

The L4 leakage profile can be described as: full plaintext under deterministic word-substitution cipher. The server learns the pattern of locations in the text where each word occurs and its total number of occurrences in the document collection [28].

The SSE systems that have a L4 leakage profile are simple legacy systems that cannot provide support for the things that an SSE system with a L1 leakage profile needs. In SSE systems with an L4 leakage profile, the client parses each input document and performs keyword extraction to produce keywords W_i that are in order with repeats. Then a deterministic cipher is applied to each word. The resulting collection of ciphertexts is then uploaded. The encrypted version of the document can be seen as an exact copy in which a cipher has been applied to every keyword. The L4 leakage profile thus shows the position of the query in the plaintext and how many times it occurs in the document. This can also be seen in the most left image in figure 3.3.

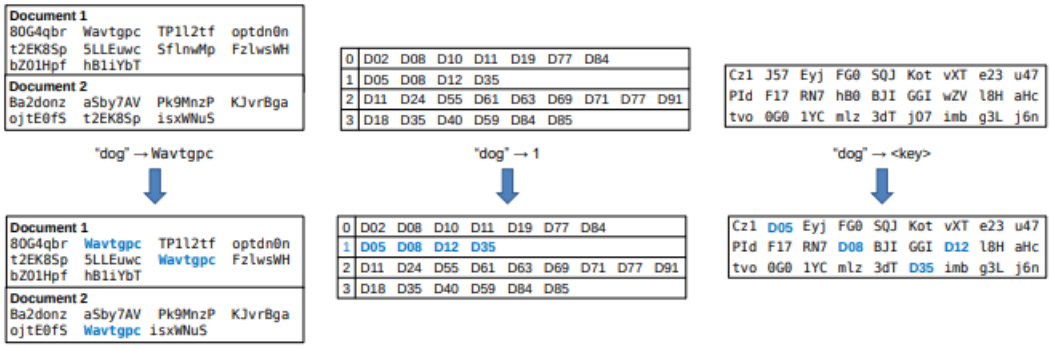


Figure 3.3: View of the server for the same plaintexts before and after searching for the 'dog' keyword for the L4, L2 and L1 leakage profiles (from left to right) [28].

3.5.2. L3 Leakage Profile

The L3 leakage profile can be described as: fully-reveled occurrence pattern with keyword order. This leakage profile fully revels the pattern of keyword occurrences in documents, in the order of their first appearance, but not the occurrence counts within a document. This leakage profile can thus be seen the same as leakage profile L4, except that in this case the repetitions are removed from the encrypted documents.

3.5.3. L2 Leakage Profile

The L2 leakage profile is similar to the L3 leakage profile. The occurrence pattern of keywords are revealed for every term, yet not in document order. Formally if the documents collectively contain terms $\cup_i W_i = \{w_1, ..., w_N\}$, then the leakage profile leaks the full collection of sets $\{i : w_1 \in W_i\}$, ..., $\{i : w_N \in W_i\}$. In less technical terms: it is an inverted index. As can be seen in the middle image 3.3, dog is the second keyword in the list and its identifier is "1". In the inverted index the document identifier of the documents that contain the keyword "dog" are listed.

3.5.4. L1 Leakage Profile

The L1 leakage profile can be described as: query-revealed occurrence pattern. This leakage profile exposes the same amount of information as the L2 leakage profile, but only for terms that have been queried. This is where the access pattern (that was described earlier in this section) comes into play, since when a query is sent, the server learns the access pattern of this query. It will be easier for the attacker to distinguish the keywords when more keywords have been queried, and thus the accuracy of the attack will be higher. The new attack that will be described later will also have the L1 leakage profile.

3.6. Attacker Models

With SSE systems, the service provider receives the encrypted dataset together with the inverted index from the client. When the service provider receives query requests, he needs to return the encrypted documents according to the inverted index. The service provider can abuse his knowledge to extract even more private information. That is why there are different attack models.

3.6.1. Honest-but-curious

In this attacker model, the attacker follows the protocol as discussed with the client, but the attacker attempts to learn as much as possible about the underlying keywords of the queries.

3.6.2. Active adversary

An active adversary is able to inject files into the SSE system. In cryptography terms this is also called a chosen-document attack. The adversary tricks the client into encrypting a chosen document. The attacker than has access to a plaintext and encrypted version of the same document.

3.7. Countermeasures

To mitigate various attacks on SSE system various countermeasures have been proposed in the literature [9, 8, 32, 37, 27, 38]. These lower the accuracy of the attacks drastically and ideally an attack would be resistant to them. The countermeasures that will be considered in this work will be padding and obfuscation. These countermeasures are proposed to mitigate known-data attacks, but they are also suitable for similar-data attacks since they alter the co-occurrence matrix that is inferred from the queries. The countermeasures have been used in for example [9].

In section 9, there will also be a subsection about how the new attack that will be shown in chapter 7 performs against these countermeasures with a comparison with other attacks as well.

3.7.1. Padding

This countermeasure works by adding in fake keyword-document pairs to the look-up table (or inverted index). This means however that the users get some false-positive documents when querying for a keyword. These can however be easily filtered when the user receives the response and decrypts the documents. There are however no false-negative results (not sending particular documents belonging to the query) because this impacts search results meaning that the user will not get the all of the documents when keyword search is performed. Because of padding the access pattern the attacker sees is not correct anymore.

3.7.2. Obfuscation

Obfuscation is a hardened form of padding. As explained with padding there can be no false-negative results. With obfuscation this is however possible. In [4], Chen et al. proposes obfuscation. In this countermeasure, documents are divided into so-called ”shards”. This allows for false-negative results because the user does not need every shard to reconstruct a document. After the shards have been computed, the obfuscation algorithm adds and removes shards from the results. False-negative shards do not result in false-negative results, but it affects the access pattern the attacker sees, and therefore it becomes harder for the attacker to see which documents where returned to the query that was being sent.

4

Related Work

Different sorts of SSE attacks exist. Most attacks are Leakage abuse attacks that use access pattern leakage, hence the whole leakage hierarchy. Known-data attacks and similar-data attacks use these leakages to predict keyword-trapdoor pairs. File Injection attacks do also exist which do not use these leakages, but work in a completely different way. This chapter discusses the different SSE attacks that currently exist at the time of writing this thesis. Only the most important papers will be discussed in detail. The attacks are divided up into three categories: Known-data attacks, Similar-data attacks and File Injection attacks. A table summing up the differences is shown in figure 4.2.

4.1. Known-data attacks

Most research about attacks on SSE systems are about known-data attacks. The attacker model is the honest-but-curious (also called passive) attack model. In these attacks, it is assumed that the attacker has access to a certain percentage of the documents stored on the SSE system. The higher the known-data rate, the higher the accuracy of the attack. However, in all cases the attacker needs to know a significant percentage of the documents stored on the SSE system to get a reasonable query recovery accuracy.

The way these attacks most of the time work is by comparing the keyword occurrence of keywords in the documents with the encrypted keyword occurrence in the encrypted documents. Since the known-data can be seen as a sample of documents from all the data, the keyword occurrences should me more or less the same as the encrypted keyword occurrences. This means that ideally it would be possible to predict keyword and encrypted keyword pairs.

Furthermore attacks also use keyword-keyword co-occurrence. This is can be seen as follows. Imagine that you have extracted all of the keywords from the document set. These keywords are the keyword universe. Now imagine you create a table in which the columns and rows are the keyword universe. Every entry in this table is now 0, except if two keywords occur in the same document. Then this entry will be the amount of documents these two keywords occur in. Now the same will be done for the encrypted keywords. The idea is now the same as with the keyword occurrence and ideally it would be possible to predict keyword and encrypted keyword pairs using keyword-keyword co-occurrence.

Different attacks use the keyword occurrence and keyword-keyword co-occurrence in different ways. The most currently known attacks will be explained now.

4.1.1. IKK attack

The IKK attack [14] is one of the first attacks on SSE systems. After the paper has come out is defined as being a known-data attack by [28], since at the time the terminology for SSE attacks was not clear yet. The attack exploits the co-occurrence pattern (or keyword-keyword co-occurrence or trapdoor-trapdoor co-occurrence) which for every two keywords (or two trapdoors) gives the amount of times they appear together in the document (or encrypted document) set. In this case the co-occurrence

pattern is a matrix in which the rows and columns are indexed by trapdoors. The entries of this matrix will be 0, except if the i'th row and the j'th column occur together in a set of documents. Then the entry will be the number of documents in this set. This is the inverted matrix, explained in chapter 3.

The IKK attack also requires a background matrix in which now the rows and columns are indexed by keywords. This is the known-data needed for the attack. In a similar fashion, all entries of this matrix are 0, except if the i'th row and j'th column appear together in a set of documents, then the entry will be the number of documents in this set. There is one difference, being that there is some added noise to this matrix. This is to simulate some of the differences between the background matrix and the co-occurrence matrix. The attack uses access pattern leakage.

The attack then works by solving an optimization problem in which it tries to find a mapping between the co-occurrence and background matrix. And this thus leads to a mapping between trapdoors and keywords. The optimization problem uses simulated annealing, but this makes the attack very computationally expensive, which in turn makes the attack take a very long time to perform the mapping. Furthermore the accuracy of this attack is also very bad. The accuracy of the attack is 0% with 5000 keywords with a 25% known-data rate with the Enron dataset.

4.1.2. Count attack

The count attack, published in [28] uses the co-occurrence pattern in conjunction with the response length pattern. Furthermore since it is a known-data attack it also requires some of the real documents. The more real documents the attacker has, the higher the accuracy of the attack will be. In addition to this, the authors of the attack assume that the server knows the number of matching documents for each keyword in the keyword universe in the real document set. The notation for this is count(w). The attack performs strictly better than the IKK attack, however the accuracy of the attack is still very low at 0% with 5000 keywords and a known-data rate of 25% with the Enron dataset, and the attack requires a lot of known-data to be somewhat effective.

The attack works as follows. The attacker has access to the encrypted documents. In addition to this, it was assumed that the attacker has access to the result count of every keyword in the keyword universe. The attacker can abuse this to find keywords with unique result counts. And since the attacker has access to the encrypted documents, the attacker can immediately predict the correct keyword belonging to a query when the result count is unique, since the attacker already has a list with all keywords and their result count. This is the first part of the attack. These mappings are guaranteed to be true.

The second part of the attack works by mapping the remaining queries to a keyword with the same response length and the same co-occurrences with respect to the recovered keywords.

4.1.3. VolAn attack

The Volume Analysis attack (or VolAn) as explained in [3] is a very simple attack that works by comparing the volume of the keywords in the known-data set with the volume of the keywords in the encrypted document set. Then it performs a volume-based analogue of frequency analysis. The accuracy is again 0% with a known-data rate of 25% with the Enron dataset.

4.1.4. SelVolAn attack

The Selective Volume Analysis attack as explained in [3] is an extension on the regular volume analysis attack. It exploits the response length pattern in addition to the total volume pattern to predict the correct keyword-trapdoor. It is a more complicated attack, but the accuracy is still very poor. The accuracy is again 0% with a known-data rate of 25% with the Enron dataset.

4.1.5. Subgraph ID/Subgraph volume attack

The subgraph attacks, published in [3] come in two sorts, the subgraph ID attack and the subgraph volume attack. The two attacks are the same, except that they use a different handle as will be discussed later.

The attack works by modeling the leakage pattern and the known-data as bi-partite graphs. The top vertices of the leakage graph are queries and the bottom vertices are the handles (the document identifier or the volumes of the documents). There is an edge between two vertices if the handle is part of the query's observed leakage.

For the known-data graph, the top vertices are keywords and the bottom vertices are handles (again document identifier or volumes of the documents). AN edge is added between the two vertices if the document contains the keyword or if the keyword has that volume.

The bottom part of both graphs are the same and the attack can then be seen as a subgraph mapping problem. The attacker tries to map the known-data graph into the leakage graph by leveraging the edge distribution of the graphs. Then there are a few filtering steps.

The attacks are the best performing known-data attacks, although they still require 25% known data of the whole document set for an accuracy of about 51% for the subgraph ID attack and 65% for the subgraph volume attack. The time complexity of the algorithm is $O(t \cdot W + \sum_{i=1}^{t} \sum_{D \in \mathbf{D}(q_i)} \#D)$, where D is a document and **D** is a document collection. W is the keyword universe, t is the amount of queries observed by the adversary and q_i is the i^{th} query.

4.1.6. SAP attack

This SAP attack [26] is one of the few attacks that use both the search pattern and the access pattern leakage. The attack tries to solve a maximum likelihood problem. With the help of a parameter the attacker can assign more weight on the search pattern or access pattern, for example if the attacker knows that their volume information is more reliable than their frequency information. The attack can be efficiently solved with the Hungarian algorithm. The attack still requires a lot of known-data (25%) and has an accuracy of about 30% with 5000 keywords with the Enron dataset.

4.2. Similar-data attacks

Similar-data attacks work in more or less the same way as known-data attacks. The attacker model of similar-data attacks is also the honest-but-curious/passive one. The difference however is the knowledge of the attackers. In known-data attacks it was assumed that the attacker has a copy of a certain percentage of the documents stored on the SSE system. Either because these were already public or because they somehow got leaked or because of something else.

With similar-data attacks however the assumption is that the attacker has created his own dataset of documents which are similar to the documents stored on the SSE system. This can for example be the case when the attacker is the SSE service provider. Imagine that a hospital wants to store their medical documents on a SSE system, which would be perfectly reasonable in this case since a hospital has thousands of documents that need encrypted before they can be stored. However the hospital staff also need to regularly search through these documents.

The SSE service provider knows that the client is a hospital because of the domain they used when signing up for the service. Therefore it is easy for the SSE service provider to assume that the client is going to store medical documents. The attacker now will gather as many medical documents as possible that are publicly available. These do not have to be any of the documents stored on the real SSE system, but the idea is that the similar documents are going to have a more or less same distribution of keywords as the real documents stored on the SSE system.

However since similar-data attacks now depend on a completely different dataset, the problem now is that leakage patterns like result length cannot be used. Also the highest achievable accuracy now is not 100%, but the (*I_size / U_size*) * 100% where I_size is the size of the intersection of the real document set and the similar document set and U_size is the size of the union of these two sets.

There are not a lot of similar-data attacks since most research has been focused on known-data attacks. In this thesis we only consider the refined score attack and not the regular score attack since the refined score attack is an improvement on the regular score attack that is discussed in the same paper. The refined score attack performs strictly better. Therefore it does not make any sense to discuss the regular version as well.

The weighted Graph Matching attacks on SC-ESE and MA-ESE systems [30] are also not discussed since these attacks are really computationally expensive. Furthermore the attack is also created for weaker SSE systems than the attacks described earlier, which are all designed for the L1 leakage profile in the case access patterns are used. It was also not possible to recreate the attack described in the paper. Nevertheless it was the first similar-data attack. An interesting note is that these attacks were focused on an SSE system that was already being used in practice: ShadowCrypt [13] and Mimesis Aegis [20].

4.2.1. Refined Score attack

The refined score attack is an iteration on the regular score attack. Both attacks are revealed in the same paper. The attack works by calculating a score for every keyword (in the similar dataset) and trapdoor (encrypted keyword). This score is calculated by taking the difference of the sub-matrix of the trapdoor-trapdoor co-occurrence matrix minus the sub-matrix of the keyword-keyword co-occurrence matrix. Then the negative natural logarithm of this value is the score. There is no particular reason for taking the negative natural logarithm other than the fact than making the scores more human comprehensible. Any other metric could have been chosen and this would not make a difference for the accuracy of the attack.

The attack works in rounds with the ref_speed parameter. Every round ref_speed keyword-trapdoor pairs are added to the predictions list.. And if there are less than ref_speed trapdoors to be predicted left the attack stops.

Furthermore the attack assumes that the adversary knows some keyword-trapdoor pairs. These are needed to project the keyword and the trapdoor co-occurrence matrices to a common sub-vector space. After all scores have been computed the keyword with the highest certainty (calculated by substracting the score of every keyword from the maximum score)is chosen as the correct keyword for the trapdoor. The keyword and trapdoor are then added to the keyword-keyword co-occurrence and trapdoor-trapdoor co-occurrence matrix respectively for the next round. The accuracy of the attack is 43% with 5000 keywords with the Enron dataset. The time complexity of the attack is $O(\frac{|Q|}{RefSpeed} \cdot |Q| \cdot m_{sim} \cdot k)$, where $|Q|$ is the amount of queries observed by the attacker, RefSpeed is the ref_speed parameter, m_{sim} is the size of the keyword universe extracted from the similar dataset and k is the amount of known trapdoor-keyword pairs by the attacker.

4.3. File Injection attacks

File injection attacks work in a completely different way than known-data attacks and similar-data attacks. The attacker model is the active adversary and as the name of the attack already suggests, the attacker injects files into the SSE system. The adversary doesn't do this directly by literally adding a file to the client's files by copying it to the same directory. This would not work since this file would not be encrypted since the adversary does not have access to the key of the client that he used to encrypt his files with. However it may work in the following way. Imagine that the SSE system is serving as an email storage server. Emails of a whole company will be stored on the SSE system when they arrive. The adversary can then 'inject' files into the SSE system by sending a mail to the company.

Since the attacker also has access to the encrypted documents, he can easily identify the file he just 'injected'. Now the attacker has a regular version of its document and an encrypted one that is encrypted in the same way as all of the other documents stored on the SSE system with the same key. In cryptography terms this is also called a chosen-document attack. The attacker can now abuse this by injecting more files to get a 100% accurate query recovery attack if he injects enough files. These files contain specific information depending on the attack. Two attacks that work solely by injecting files exist.

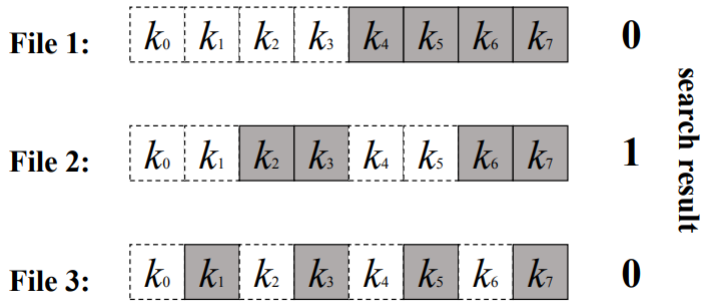


Figure 4.1: An example of the binary-search attack [40]

The way they work and how they achieve query recovery will be explained in the next subsections.

4.3.1. Binary search attack

This binary search attack is a relatively simple attack explained in [40] works by first generating log|K| files, where K is the size of the keyword universe. These files are then listed in arbitrary order (it does not matter in which order these words are listed), and in files 1 to log|K| we only put the words whose i^{th} bit is 1, we now have log|K| files containing the entire keyword universe. These are the files that need to be injected into the SSE system. The content of these files are the keywords the attacker wishes to get the trapdoor of. In a full attack the adversary injects a lot of files containing the entire keyword universe and this also results in a 100% accuracy. A smaller universe binary search attack is also possible, this way fewer files are injected and the attacker only learns the keyword-trapdoor pairs of the keywords that are injected.

The query recovery part of the attack then works as follows. If the user now queries for a particular keyword, among the regular files that should have been returned some (or none at all) of the files will be returned. According to which files are returned, the attacker can now determine which of the keywords belong to the requested trapdoor.

As an example we can take the situation as explained in [40]. We have 8 keywords and the log of that is 3, meaning we need to inject three files. We can see the distributions of the keywords in figure 4.1. The files contain the shaded keywords. We can clearly see that if file 2 and file 3 are returned to the user, the requested keyword should be k_3, since this is the intersection of these two files. If none of the files are returned the keyword should be k_0, since none of these files contain k_0. If only 1 of the files i returned, lets say file 1, than the requested keyword is k_4.

In [3] the authors also publicize a binary search attack, which is more or less the same as the one as described in this section but uses the volume instead.

Binary search attack using known-data

The problem with the binary search attack is that we need to inject log|K| files and that if the keyword universe is very large, a lot of files need to be injected, which makes the chance of being detected larger. Therefore the authors in [40] also decided to use known-data together with the binary search attack to decrease the amount of files that need to be injected into the SSE system. Unfortunately this also causes the accuracy of the the attack to drop significantly. Only with a lot of known-data the accuracy remains reasonable. But this idea sheds light to another possibility for other attacks.

Attack	Leakage pattern	Required information	Accuracy
IKK	Access pattern	known-data, known queries	0%
Count	Access pattern + response length pattern	known-data, known queries	0%
VolAn	Total volume pattern	known-data	0%
SelVolAn	Total volume pattern + response length pattern	known-data	0%
Subgraph-ID	Access pattern	known-data	51%
Subgraph-VL	Volume pattern	known-data	65%
SAP	Access pattern + Search pattern	known-data	30%
Refined score	Access pattern	similar-data + known keyword-trapdoor pairs	43%
Decoding	Total Volume pattern	Volume of documents of all keywords + file injection	100%
Binary search	-	All keywords + file injection	100%

Figure 4.2: A comparison on the different attacks described in this chapter.

4.3.2. Decoding attack

The decoding attack as explained in [3] works by first analyzing the volume of all keywords. Then the adversary calculates an offset which is used to make the volumes of all keywords unique. This means that files are created with a certain volume for a particular keyword to make the volume of the documents of that keyword unique. These files are then injected into the SSE system. Now the attacker can easily predict the correct keyword-trapdoor pair when a user queries a particular keyword since he can calculate the volume of the returned documents. By looking up which keyword has which particular volume, he can do the right prediction.

5

Research Question

The main research question of this thesis is:

What is the current state of similar data query recovery attacks and is it possible to create a new and better attack?

This is the main research question will be answered by decomposing it into sub-questions. Eventually the answers to this sub-questions, in combination with some experiments, will lead to the answer of the main research question. The main research question will be decomposed into the following sub-questions:

- SQ1: What is currently the best similar data query recovery attack?
  - This sub-question aims to discover what currently is the best similar-data query recovery attack. Since there are only two papers researching these it will be a small comparison to see which of the attacks is better. This attack will then become the basis for the new attack.
- SQ2: How would one improve the best similar-data attack?
  - This sub-question aims to improve the best similar-data query recovery attack by assessing the weaknesses of the attack. These weaknesses will then be improved (where possible), to in the end create a better more accurate attack for query recovery.
- SQ3: What would a similar-data attack together with a file injection attack look like?
  - In this sub-question, the best similar-data query recovery attack will be combined with a file injection attack to see what impact this has on the accuracy of the attack. The idea is similar to the partial known-data partial file injection attack in [40].
- SQ4: Will the new attack be resistant to countermeasures?
  - After improving the best similar-data query recovery attack, this sub-question will look how the new attack performs against countermeasures (obfuscation and padding). There will also be a comparison between other attacks on how they perform against the same countermeasures.

The reason for why this thesis will focus primarily on similar data attacks is because currently not a lot of research has been focused on similar-data attacks. As shown in chapter 4 only two papers have put efforts into creating similar-data attacks.

Furthermore similar-data attacks are also more realistic than known-data attacks, since as shown in chapter 4 all of the known-data attacks require a lot of known-data to have a reasonable accuracy. This known-data rate is atleast 25% of the whole dataset which makes these attacks so unrealistic, since in a 30K document dataset it would mean that the attacker already has 7.5K of these documents.

On the other hand with similar-data attacks which makes them very realistic is that the attacker only has to know the general subject of the documents stored on the SSE systems. This can be easily done by the attacker, if the attacker is the SSE service provider. A simple email or name when signing up for the service is more than enough for the SSE service provider to discover what sorts of documents are stored on the SSE system by simply googling the email or name to see where the user is working. This might for example be a hospital meaning that the SSE system might be used for storing medical documents. The similar data of the attacker does not have to include any of the documents stored in the SSE system. And this is what makes them so realistic. The attacker can build his own similar dataset.

Assessing whether or not similar-data attacks are really more realistic than known-data attacks is a topic for another thesis but this small analysis shows that similar-data attacks are currently more realistic.

6

Deeper look into the Refined Score attack

The refined score attack has already been explained in chapter 4. However since it is one of the few similar-data attacks and since this thesis will focus on creating an improved similar-data attack, the attack will be explained in more detail. As explained earlier, the Weighted Graph Matching attacks in [30] will not be considered, since they are designed for weaker SSE systems, whereas the refined score attack is designed for an SSE system with the strongest L1 leakage profile. At the time of writing this thesis only three similar-data attacks were known. The refined score attack which will be discusses now, the score attack which is a previous version of the refined score attack which is strictly worse, and the weighted graph matching attacks from [30]. Since the refined score attack is the best of the 3, it is the attack that will be chosen to be improved in this thesis. The pseudocode of the refined score attack [9] is shown below.

The refined score attack works in three steps as indicated by the comments in the pseudocode. As the name of the attack suggests, it is a refinement on the score attack. The difference being that the score attack did not have any RefSpeed parameter and that the regular score attack only assigned the keyword with the highest score to the trapdoor The score attack also did not have the known trapdoor-keyword pairs assumption. Its accuracy was also lower.

The first step of the refined score attack is simple. Since the attack works with a loop, in each iteration the trapdoors of the predicted trapdoor-keyword pairs are subtracted from the unknown queries. In the first iteration of the loop the *KnownQ* list contains the keyword-trapdoor pairs that are assumed to be known by the attacker. The code of this attack randomly assigns some keyword-trapdoor pairs to this list. This is due to the fact to make the attack more realistic since it is very unreasonable to assume that the attacker only knows one certain trapdoor-keyword pair set.

Even though the authors of [9] tried to keep it realistic by assigning a random trapdoor-keyword pair set in each attack, it is still very unlikely for an attacker to have a list of some known trapdoor-keyword pairs, even though this list can be very small for a reasonable accuracy. The attackers do not give an example of how the attackers might have acquired these trapdoor-keyword pairs. It would be more reasonable if the attack derived this trapdoor-keyword pair set some way. Furthermore, since this list is created with random trapdoor-keyword pairs and since the keyword keyword co-occurrence sub-matrix and trapdoor trapdoor co-occurrence sub-matrix are built upon these known trapdoor-keyword pairs, they can lower the accuracy of the attack, since we use these matrices to give a score to keyword-trapdoor pairs. If this score is calculated with not very accurate keyword-trapdoor pairs, the accuracy of the attack will be lower.

To give an example, let's say that one of the known trapdoor-keyword pairs is telephone-enohpelet. This is not actually the case since there is no case of encryption but it serves as an example, since the attacker does not know the literal encrypted version of the word "telephone". Since a random trapdoor-keyword pair is chosen and since the real document set and similar document set are not one-to-one copies, it might be the case that the word telephone in the similar document set occurs way more often than the encrypted word "enohpelet" in the real encrypted document set. This causes the keyword-keyword co-occurrence table to be not as accurate from the beginning of the attack and this leads to a lower accuracy through the rounds of the attack. Therefore it is of importance that the attacker has acquired frequently occurring keyword-trapdoor pairs, such that the attack can be more accurate.

In the second part of the attack, a score for every keyword in the similar dataset keyword universe and every trapdoor is calculated. A candidate list is created and a score is calculated according to the co-occurrence sub-matrices. The problem with this part is however that it only depends on the co-occurrences of the keywords and trapdoors. Perhaps the keyword and trapdoor occurrence could have been used as a filter to make the attack more accurate. Especially since the keyword occurrences are already calculated for the keyword-keyword co-occurrence. Ideally, in a correct trapdoor-keyword pair, both trapdoor and keyword would have a close occurrence in their respective real and similar data set, if the similar dataset is good enough. This means that some assumptions can be made by the attacker, since in an improved attack the occurrence of keywords and trapdoor could be useful to eliminate some of the pairs.

Finally in the third part of the attack there is another flaw. The attack stops before having made a prediction to all trapdoors, since the stopping condition is $|unknownQ| < RefSpeed$. This means that for at most RefSpeed queries, no predictions are made, which lowers the accuracy, since if there are less than RefSpeed trapdoors the attack stops. The attack could continue and this can be enforced by setting the RefSpeed parameter to 1, but this would make the attack take a very long time, since only the most certain prediction is added to the final_pred list in each iteration of the loop. Also the use of a parameter makes it hard for the attacker to perform the attack since the attack has to be run multiple times to determine the most optimal value for this parameter. This is because higher values make the attack run faster, but less accurate, and lower values make the attack more accurate, but much slower.

Improving these three flaws could drastically improve the accuracy of the attack. In the next section an improvement on the refined score attack will be discussed in which these three flaws will be remediated.

Algorithm 1 Pseudocode for the Refined Score Attack

Require: K_{sim}, C^s_{kw}, C^s_{td}, Q, KnownQ, RefSpeed
final_pred $\leftarrow$ []
unknownQ $\leftarrow$ Q
while $unknownQ \neq \varnothing$ **do**
  %1. Extract the remaining unknown queries
  unknownQ $\leftarrow \{td : (td \in Q) \wedge (\nexists kw \in K_{sim} : (td, kw) \in KnownQ)\}$
  temp_pred $\leftarrow$ []

  %2. Propose a prediction for each unknown query
  for all $td \in$ unknownQ **do**
    $cand \leftarrow$ [] ▷ The candidates for trapdoor td
    for all $kw \in K_{sim}$ **do**
      s$\leftarrow -ln(||C^s_{kw}[kw] - C^s_{td}[td]||)$
      append $\{$"kw" : kw, "$score$" : $s\}$ to $cand$

    sort $cand$ in descending order according to score.
    certainty $\leftarrow$ score($cand[0]$) - score($cand[1]$)
    append $td, kw(cand[0]), certainty)$ to $temp\_pred$

  %3. Either stop the algorithm or keep refining.
  if $|unknownQ| < RefSpeed$ **then**
    final_pred $\leftarrow$ KnownQ $\cup$ *temp_pred*
  else
    Append the RefSpeed most certain predictions from *temp_pred* to KnownQ
    Add the columns corresponding to the new known queries to $C^s_{kw}[kw]$ and $C^s_{td}[td]$

return *final_pred*

7

New Attack

In the previous section three flaws of the refined score attack were shown. In this section those flaws will be remediated to create a better performing attack in terms of accuracy. Each flaw will have its own subsection to discuss how it is remediated.

7.1. Improving Known Queries assumption

As explained in the previous section, it is very unlikely for an attacker to have acquired a set of trapdoor-keyword pairs. However, this part is essential to the attack since these known query-trapdoor pairs are needed to build the co-occurrence sub-matrix. Plainly assuming that the attacker has access to these pairs can be replaced by another assumption which is perhaps more realistic. If we assume that the attacker can inject files into the SSE system, we can drop the initial assumption. The refined score attack (a similar-data attack) can thus be combined with a file injection attack. The reason for using a partial similar-data attack and a partial file injection attack is that the risk of getting caught gets higher when more files (and thus more keywords) are injected to the system. Thus an attacker would ideally inject as few files as possible. These first few pairs that the attacker learns by injecting files can be referred to as ”base-mapping”. After the base-mappings, the attacker can focus on correctly predicting the rest of the trapdoor-keyword pairs.

In the binary search attack [40] for example, log|N| files need to be injected where N is the size of the keyword universe. If we assume that the size of the keyword universe is 4096 keywords, then 12 files need to be injected. However in the case of a smaller keyword universe binary search attack this number obviously drops since fewer files need to be injected. However this thus also means that the accuracy of the attack drops. But a smaller keyword universe binary search attack could be useful to create some base-mappings.

Since only a small number of known query-trapdoor pairs are needed for the refined score attack to have a reasonable accuracy, using a small universe binary search attack for the known queries is a viable solution. Therefore in the new attack, the assumption of the attacker knowing a few trapdoor-keyword pairs has been replaced by the assumption that the attacker can inject files into the SSE system. This assumption is more likely, since the attacker can in this case be the SSE service provider, and as explained previously, the SSE system can serve as an email storage server, meaning the SSE service provider can inject files into the system by mailing the client a certain keyword set a few times (for each file that needs to be injected).

The way a binary search attack is executed by the new attack is as follows. The attacker sorts the keywords in his similar document set according to occurrence. The assumption is then that these keywords also occur in the keyword document set of the real documents. Then, depending on how many keywords the attacker wants to inject, $\lceil \log|n| \rceil$ files are injected, where n is the amount of keywords the attacker wants to inject. Ideally this number n would be a power of two, such that no space of the injected documents is wasted.

After these files have been injected, the small keyword universe binary search attack is performed as described in chapter 4. Due to the nature of this attack, the accuracy of this part of the attack is always 100% for the keywords that are injected. Therefore the attacker now has a set of known trapdoor-keyword pairs. This set now serves the same purpose as the known query-trapdoor pairs before for the rest of the attack. Only the way the attacker got access to these pairs has changed. Instead of it being an assumption, there is now a real attack taking place.

Furthermore as you (the reader) might have noticed, only the most frequently occurring keywords are injected and assumed to be now known by the attacker, whereas with the refined score attack a random trapdoor-keyword pair list was assumed to be known by the attacker. This has the following advantage: the co-occurrence sub-matrix that will be built, will be more accurate. The reason for this can be derived as follows: with the refined score attack, a random set of query-trapdoor pairs are chosen from the keyword universe and this might mean that pairs are chosen that do not occur a lot of times in the document set. This has the disadvantage that the number of times the keyword occurs in the similar document set and the amount of times the trapdoor occurs in the encrypted real document set might be very different from each other. For example if the keyword occurs 20 times in the similar document set and the query occurs 10 times in the real encrypted document set. This is a difference of factor 2. However if we take a trapdoor-keyword pair that occurs very frequently in the document sets, and lets say that the difference is this case is also 10, just as before, then the factor difference is much smaller. Therefore a more accurate co-occurrence sub-matrix can be built in the beginning, which leads to an attack with a higher accuracy, since the scores are calculated with the help of these sub-matrices.

7.2. Use of keyword co-occurrence

As explained in the previous section, the refined score attack does not use the keyword occurrences as part of the attack. This is a missed chance, since the occurrences are already calculated since they are needed to compute the keyword-keyword co-occurrences. Ideally one would also create a score for the keyword occurrence and combine this with the already existing score of the refined score attack. However after many tries, the author was not able to create an attack with a better accuracy this way.

Instead a more simple approach is to use the keyword occurrence as a very simple filter. Remember that for every trapdoor a score is calculated with every keyword. So there is a list with keywords and their scores. This list is then altered as follows. If the keyword in the similar dataset occurs more than twice as often as the trapdoor in the encrypted real document set (or if the trapdoor occurs more than twice than the keyword), the keyword is dropped from the score list, meaning that it will not be considered anymore for the prediction of the trapdoor.

The idea behind this approach is follows. Ideally if the attacker has an accurate similar data set, the keyword occurrences in the real document set will be more or less the same as the keyword occurrences in the similar dataset. If however there is a factor 2 difference, this is probably not the case. Therefore, even though if these two keywords have a similar keyword-keyword co-occurrence, because their keyword occurrences differ a lot, they are dropped and not being considered as a prediction.

7.3. Removing use of RefSpeed parameter

Finally remember that the refined score attack used the RefSpeed parameter and that if there were less than RefSpeed trapdoors that still needed to be predicted the attack stopped. The new attack now only stops when there is a prediction made for *all* trapdoors. This increases the accuracy of the attack a little bit. Furthermore , with the new attack, instead of using the most certain RefSpeed trapdoor-keyword pairs as a prediction, only the top 0.5% most certain trapdoor-keyword pairs are used as a prediction in each iteration of the loop. This also makes the attack a little bit more scalable, since if there are more trapdoor-keyword pairs, a larger amount of trapdoor-keyword pairs will be taken as a prediction in each round. With a smaller amount of trapdoor-keyword pairs the attack is also more accurate but it takes more time to execute, since in each round fewer trapdoor-keyword pairs are added to the prediction list, but these pairs have a higher chance to be correct. Another advantage is that the attacker now

does not have to set a value for this parameter.

7.4. New attack

Overall this new attack can be seen as a major improvement on the refined score attack. The idea of the refined score attack was already really powerful, however as explained in the previous section and as shown in the current section, it needed some improvements. The new attack is now a similar-data file injection attacks, and to date it is the first attack to do so (to the best of the authors knowledge). The pseudocode of the attack can be found below. The next section will show how the experiments will be conducted.

Algorithm 2 The New Attack

Require: K_{sim}, C^s_{kw}, C^s_{td}, Q, f
final_pred $\leftarrow$ []
unknownQ $\leftarrow$ Q

%1. Perform small universe binary search file injection attack
$to\_inject \leftarrow$ most frequent f keywords in K_{sim}
Inject to_inject into SSE system
KnownQ $\leftarrow$ keyword-trapdoor pairs of to_inject after injection

while $unknownQ \neq \varnothing$ **do**
  %2. Propose a prediction for each unknown query
  for all $td \in$ unknownQ **do**
    $cand \leftarrow$ [] ▷ The candidates for trapdoor td
    for all $kw \in K_{sim}$ **do**
      s$\leftarrow -ln(||C^s_{kw}[kw] - C^s_{td}[td]||)$
      append $\{"kw" : kw, "score" : s\}$ to $cand$

    sort $cand$ in descending order according to score.
    certainty $\leftarrow$ score($cand[0]$) - score($cand[1]$)

    for all $kw \in K_{sim}$ **do**
      if $kw\_frequency/td\_frequency > 2$ **or** $kw\_frequency/td\_frequency < 0.5$ **then**
        remove kw from $cand$

    append($td, kw(cand[0]), certainty$) to $temp\_pred$

  %3. Either stop the algorithm or keep refining.
  if $unknownQ = \varnothing$ **then**
    final_pred $\leftarrow$ KnownQ $\cup$ *temp_pred*
  else
    Append the top 0.5% of the most certain predictions from *temp_pred* to KnownQ
    Add the columns corresponding to the new known queries to $C^s_{kw}[kw]$ and $C^s_{td}[td]$

return *final_pred*

8

Experiments

This chapter will contain some information regarding the experiments that will be done in order to compare the new attack with the already existing attacks. To validate the results, three different datasets will be used. Furthermore, the way the attacks will be executed will also be discussed.

8.1. Datasets

Three different datasets will be used to evaluate the performance of the attacks. These datasets will be the Enron dataset, the Apache Lucene dataset and finally the Wikipedia dataset.

8.1.1. Enron

The Enron Corpus is a dataset of over 600.000 emails generated by over 150 employees of the Enron corporation [10]. It is publicly available online and is one of the few datasets that contains real emails. The Enron dataset is widely used in the literature of SSE attacks as well as in other various studies. The Enron corpus consists of a set of folders for each employee containing all of the emails of that employee, either sent or received. The literature mostly focuses on the *_sent_mail* folder since this folder contains real emails written by the employees themselves. As explained before, one of the real uses of an SSE system, could be to serve as an email storage server.

In the experiments that will be conducted, only the emails in the *_sent_mail* folder will be considered. This amounts to 30109 emails, which is more than plenty for the experiments. Before these emails can be used though, there is the need for some preprocessing of each email, due to the fact that the first few lines of every email contains some meta-data like Message-ID, Date, Content-Type etc. These lines will be removed. An example of a raw email can be found in appendix A and of a preprocessed one can be seen below.

The preprocessed email from allen-p's _sent_mail folder, number 13:

```
Jim,
Is there going to be a conference call or some type of weekly meeting about all the
regulatory issues facing California this week?  Can you make sure the gas desk
is included.
Phillip
```

8.1.2. Apache Lucene

The second dataset that will be used is the Apache mailing list archives [24]. Of these mailing list the "java-user" mailing list from the Lucene project of the years 2002-2011 will be used. This dataset has also been used in SSE attack literature as for example in [28, 9, 26, 2]. The dataset contains 50878 documents. These emails also require some preprocessing. Since this is a mailing list, the subscribers might want to opt out. Therefore a line has been added to each email to unsubscribe from the mailing list. Preprocessing is needed to remove the "To unsubscribe" line from every email, since technically it

is not part of the email. Furthermore, each email, (similar to the emails in the Enron dataset), contain a lot of metadata. This also needs to be removed. An example of a raw email can be found in appendix B and a preprocessed email can be found below.

The reason for using this dataset is the same as for using the Enron dataset. An SSE system might be used for email storage and the documents in this dataset serve as real emails.
The preprocessed version of the Apache Lucene dataset from the mailing list of December 2009:

```
I'm upgrading from 2.3.1 to 3.0.0. I have 3.0.0 index readers ready to go
into production and writers in the process of upgrading to 3.0.0.

I think understand the implications of
http://wiki.apache.org/lucene-java/BackwardsCompatibility#File_Formats for
the upgrade, but I'd love it if someone could validate my following
assumptions.

1. My 2.3.1 indexes have compressed fields in them, which the 3.0.0
readers work nicely with, as expected. I should assume that my 3.0.0 readers
will continue to handle 2.3.1 indexes OK.

2. Presumably Lucene all future 3.x index readers will continue to handle
compressed fields and we should only anticipate Lucene 4.x choking on them.

I was naively expecting my index directories to grow when my 3.0.0 index
writer merged the 2.3.1 indexes and/or optimize()'d them converting them to
3.0.0. However, I don't see that. Presumably that means that....

3. Documents added to existing 2.3.1 indexes will be added conforming to
3.0.0, but existing documents in the index will continue to have compressed
content and old documents can coexist happily with the new ones, and my
indexes will become a mixture of 2.3.1 and 3.0.0.

4. I should use
http://lucene.apache.org/java/2_9_1/api/all/org/apache/lucene/util/Version.h
tml#LUCENE_23 for the StandardAnalyzer and QueryParser in mixed indexes in
3.0.0 if I want to handle analysis consistently, or go for LUCENE_CURRENT if
I want to handle the new content "better" (bearing in mind that the new
content will eventually replace the old content anyhow).

5. I should use
http://lucene.apache.org/java/3_0_0/api/all/org/apache/lucene/analysis/StopF
ilter.html#StopFilter%28boolean,%20org.apache.lucene.analysis.TokenStream,%2
0java.util.Set%29 with enablePositionIncrements=false in mixed indexes in
3.0.0 if I want to handle analysis consistently, or go for
enablePositionIncrements=true if I want to handle the new content "better"
(bearing in mind that the new content will eventually replace the old
content anyhow).
```

8.1.3. Wikipedia

The third dataset that will be used is a Wikipedia dataset. The reason for this is that Wikipedia is a website that hosts all kind of information about all kind of different subjects, meaning that the keyword universe in this case is much larger and that the documents will all vary very much from each other. The Wikipedia dataset that will be used is the simplewiki dump of the 20th of August[1]. The simplewiki dump is a dump of the Simple English Wikipedia [34], which is a smaller subset of Wikipedia pages for everyone including children and adults learning English in which the articles are written in simple

[1] https://dumps.wikimedia.org/simplewiki/20220820/

English. The dump consists of around 215k documents. Unpacked it is around 1GB whereas the whole Wikipedia dataset is 80GB in size which would be too much to conduct experiments with during a thesis.

The simplewiki dump contains around 215k documents. The documents in this dataset also require some preprocessing, as the dump is only a single xml file. Therefore a special tool on GitHub [33] has been used to convert this xml file into plaintext files in the form of JSON. This JSON file is created for every page in the dataset and has three key-value pairs: id, text and title. For the experiments only the text key-value pair is needed. Furthermore there are random *'s in the documents that also need to be removed. An example of one of the raw documents can be seen below:

```
{
 "id": "58",
 "text": "The word application has several uses. *In medicine, 'application' means
 putting some drug or ointment usually on the skin where it is absorbed into the
 human body. *In computer software, an application is a type of program which is
 designed for a particular function. Example: word processing. *In business or
 government, an application is a (usually paper) form filled out and handed in by
 a person seeking a privilege from a state or company, such as work, credit, some
 type of license or permit, or a place to live. *At work, generally engineering,
 when dealing with certain materials or objects, an \"application\" is a purpose
 that material or object can be used for. Wood and steel have many applications.",
 "title": "Application"
}
```

8.1.4. Comparison

Since three different datasets will be used, and the results of the attack may differ from dataset to dataset, a comparison between the datasets will be made in this subsection, since this might allow to make a correlation between the difference in results and difference between the datasets. To show a comparison between the datasets, a few statistics are needed. The following statistics will be evaluated and the results can be seen in the table below. The statistics that will be evaluated are: average length of documents (in words), amount of words in longest document, size of the keyword universe of all documents, average size of keyword universe per document and finally the amount of documents. These numbers are calculated after preoprocessing, since the attacks will also be performed on the documents after preprocessing (and after stemming and removing stopwords as will be explained later.

	Enron	Apache	Wikipedia
Avg. length per document	88.88	130.30	124.95
Length of longest document	14167	14040	28152
Size of the keyword universe	63029	92401	645550
Avg. keyword universe per document	57.31	77.93	71.04
Amount of documents	30109	50878	214596

Figure 8.1: A table showing some statistics of each dataset

As can be seen from the table, the average length of the documents in the Apache and Wikipedia dataset are much larger than the average length of the documents in the Enron dataset. Furthermore the longest document in the Wikipedia dataset is twice as long as the longest documents in the Enron and Apache datasets. The Enron dataset also has the smallest keyword universe, and the Wikipedia dataset has a (more than) 10 times larger keyword universe. However the average size of the keyword universe per document is larger for the Apache dataset, even though the total size of the keyword universe is much smaller. This suggests that the Wikipedia dataset contains a much more diverse set of documents about different topics. This is also very reasonable since Wikipedia contains information about all sorts of different topics. And as previously shown, the size of the document set of Wikipedia

is much larger than the other two datasets.

8.2. Natural language processing of keywords and queries

Since this thesis is interested into query recovery (as are almost all papers about attacks on SSE systems), keyword extraction has to be performed on all documents. Keyword extraction is performed on the preprocessed documents. Since words come in multiple forms (for example walk/walking), the keywords have to be stemmed. All papers in the literature on attacks on SSE systems use Porter Stemming [36] and that will also be the case for this thesis. The Porter stemmer of the NLTK library [23] for Python is used. Furthermore the natural English language also contains a lot of stopwords (examples are: a, from, me, i). These need to be removed, since the frequency of these keywords are very high compared to other words. Furthermore the chance that these will be queried by a client is also very small since a very big subset of the documents will be returned when this is done. Again, the NLTK library for Python is used to remove these stopwords.

Since there is no real-world query set on the Enron, Apache Lucene and wikipedia dataset or way to derive query patterns of individual users publicly online, the queries will be randomly chosen from the keyword set of the similar document set. Meaning that each keyword might be able to be a query of a user.

8.3. Technical framework

The attack is built upon the code of the refined score attack codebase. The improvements that were discussed in the previous chapter are thus implemented directly to the already existing source code of the refined score attack. This has the advantage that most of the code for reading files, generating keywords etc. has already been written, such that most work can be put into researching a new attack. Another advantage is that the code is already highly parallelized, which makes the attack run faster.

The attacks will be run as follows: each run the document corpus of the selected dataset is split into two halves. One represents the similar dataset and the other represents the real dataset. The attacker only has access to the similar document set. Furthermore it is important to note that these splits are disjoint and thus non-overlapping. Afterwards keyword are extracted and the natural language processing techniques described in the previous subsection are applied. The results that will be shown in the next chapter, will be an average of 20 runs. In each run a random split will be made of the document corpus, such that in each run the similar and real document set will be different.

After keyword extraction has been performed on the real and similar document set, a random subset of the keyword universe of the real document set will be chosen to serve as queries (or trapdoors). These queries are encrypted and the attack will try to find a correct keyword in his similar document set for this query. The accuracy of this attack is then calculated as correctly_predicted_queries / total_queries * 100%. In this equation correctly_predicted_queries are the queries that the attack predicted correctly, thus for the queries that it found the correct keyword for. This includes the first few trapdoor-keyword pairs that are injected into the SSE system, since the attackre did not know these pairs before.

8.4. Hardware

All experiments are done on the authors laptop. The laptop is running on Arch (a Linux distribution). The CPU is a 8-core AMD Ryzen 7 5800HS with a total of 16 threads and 16GB of ram.

8.5. Experiments to be conducted

The refined score attack and the new attack have a few settings that can be set for each experiment. Obviously the first one is the chosen dataset. Furthermore the size of the vocabulary from the similar document set and the size of the queryable vocabulary (vocabulary of real document set) can also be set. These numbers are always set to the same value. The number of queries that are observed by the attacker can also be set. The accuracy of the attack is based upon this number. For example if 500 queries are observed and the attack predicted 250 of these correctly, than the accuracy of the attack

is 50%.

The keywords that will be chosen from the similar document set and from the real document set will be the most frequently occurring keywords from each of the datasets. This is referred to as high selectivity in the literature [3]. Almost all attacks in the literature are performed with this setting. The observed queries are than randomly sampled from the queryable vocabulary.

Finally, the amount of known queries by the attacker can be set. For the new attack however this parameter sets the amount of keywords that will be injected into the SSE system. Obviously, the higher this number is, the higher the accuracy will be of the attack, since a more accurate keyword-keyword co-occurrence sub-matrix will be created from the beginning as described in chapter 6 and chapter 7. Obviously in the case of the refined score attack the known trapdoor-keyword pairs makes the attack less realistic.

The new attack will be compared to the refined score attack, and to the Subgraph-VL attack. The known-data rate for this attack will be set at 3.33%, which is more than reasonable, since it is very unlikely for an attacker to already have access to more than 1000 documents of a 30109 document set (in the Enron case).

In the experiments there will be three different settings that will be considered. And for each of these three settings the attacks will be conducted with each of the three datasets, except for the Wikipedia dataset. Because the Wikipedia dataset has a very large keyword universe (see figure 8.1, it is not possible to conduct the third attack setting with the large keyword set. The problem is that the co-occurrence and occurrence matrices are too large to fit into RAM, because of the huge size of the keyword universe. Therefore only the small and medium keyword set attack settings will be conducted on the Wikipedia dataset. Furthermore for each attack scenario (attack setting and dataset), the refined score attack and the new attack will also be conducted with the padding and obfuscation countermeasures.

The three attack settings are:

- Small keyword set: in this setting there will be 1000 keywords in the real and similar keyword set (also 1000 keywords being considered for Subgraph-VL), with 150 trapdoors being considered. Furthermore for the refined score attack 4 known keyword-trapdoor pairs will be assumed and 4 keywords will be injected in the case of the new attack (and thus 2 files).
- Medium keyword set: in this setting there will be 3000 keywords in the real and similar keyword set (also 3000 keywords being considered for Subgraph-VL), with 450 trapdoors being considered. Furthermore for the refined score attack 8 known keyword-trapdoor pairs will be assumed and 8 keywords will be injected in the case of the new attack (and thus 3 files).
- Large keyword set: in this setting there will be 5000 keywords in the real and similar keyword set (also 5000 keywords being considered for Subgraph-VL), with 500 trapdoors being considered. Furthermore for the refined score attack 16 known keyword-trapdoor pairs will be assumed and 16 keywords will be injected in the case of the new attack (and thus 4 files).

8.6. Evaluation

As explained previously, each attack scenario will be run 20 times, and the average accuracy of these 20 runs determines the accuracy of the attack. Thus in the evaluation part the most important thing, when we compare the new attack to the refined score attack and to the Subgraph-VL attack is that this average accuracy is higher.

9 Results

9.1. No countermeasures

9.1.1. Enron

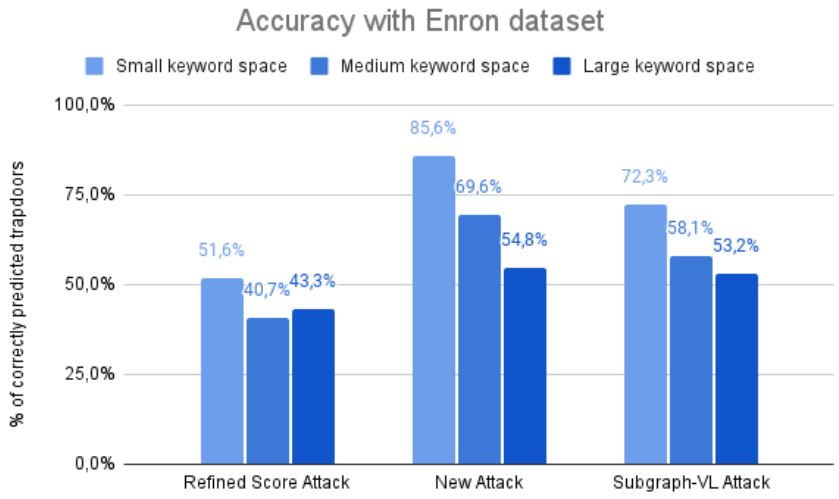


Figure 9.1: Accuracy of the refined score attack, new attack and the subgraph-VL attack with the Enron dataset

As can be seen from figure 9.1, the accuracy of the new attack is better than the accuracy of the refined score attack and the subgraph-VL attack in each of the three attack settings. Furthermore what can be seen from the results in this graph (and all other graphs) is that the accuracy drops as the keyword space increases. The reason for this is that as the keyword space increases, the amount of possible keyword-trapdoor pairs also increases, meaning that more keywords might have the chance of being paired with a trapdoor in a prediction, since their occurrence and co-occurrence might be more similar. Very small differences between keyword-keyword co-occurrences and keyword occurrences could make the difference for a prediction and hence it becomes harder to predict the correct keyword-trapdoor pair and the accuracy thus decreases.

9.1.2. Apache Lucene

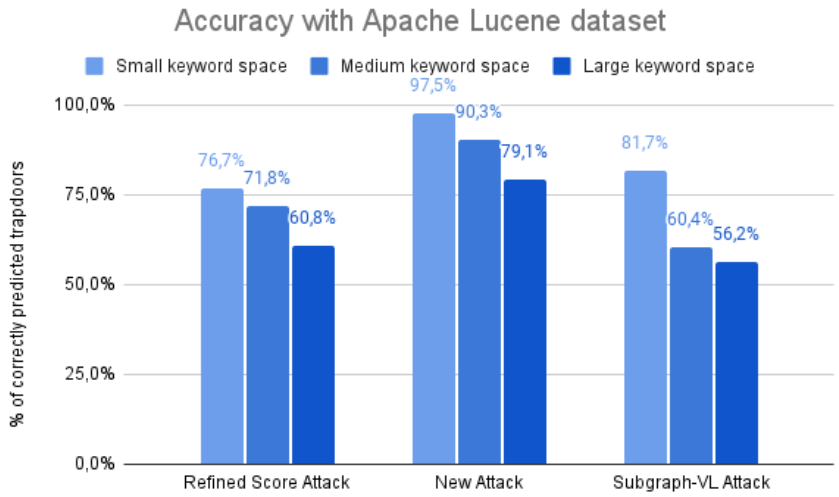


Figure 9.2: Accuracy of the refined score attack, new attack and the subgraph-VL attack with the Apache dataset

As can be seen from figure 9.2, the accuracy of the new attack is still better than the refined score attack and subgraph-VL attack in each setting. As was the case with the Enron dataset, the accuracy drops as the keyword space increases. The accuracy of the refined score attack and subgraph-VL attack is still very high, but the accuracy of the new attack is still better, almost reaching an accuracy of 100% for this dataset.

9.1.3. Wikipedia

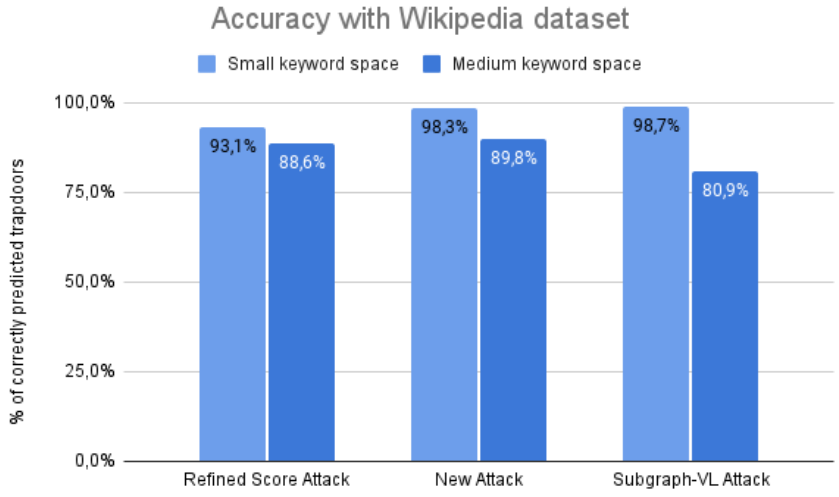


Figure 9.3: Accuracy of the refined score attack, new attack and the subgraph-VL attack with the Wikipedia dataset

With the Wikipedia dataset the attacks have only been conducted with the small and medium keyword sets. The reason for this is that the Wikipedia dataset is very large, and therefore it is not possible to fit the occurrence and co-occurrence matrices into the RAM of the machine running the attacks, because of the huge keyword universe and document set.

The accuracy of each attack is very high with this dataset. The reason for this could be the enormous size of the keyword universe. The subgraph-VL attack is only 0.4% better than the new attack with the small keyword universe. But in the medium keyword universe the new attack is 8.9% better than the subgraph-VL attack. Bear in mind that this is with 3.3% known-data and that with the Wikipedia dataset, this means that attacker has access to more than 7000 plaintexts of the document set.

9.1.4. Comparison

When looking at the results of each of the three datasets, the new attack performs better in (almost) all cases than the refined score attack and subgraph-VL attack. The average accuracy in the Apache Lucene and especially in the Wikipedia dataset is very high. This is perhaps due to the fact that these two datasets have a much larger keyword space (especially the Wikipedia dataset which has the largest keyword space and therefore highest average accuracy). This is just a mere correlation between the results and the size of the keyword space, but it could also be a causation, the reason being that since the keyword space is larger, each keyword might have a more distinct keyword occurrence and keyword-keyword co-occurrence, making it easier for the attacker to correctly predict keyword-trapdoor pairs. More research is needed to confirm this, but the average accuracy getting higher with the Apache Lucene dataset and even higher with the Wikipedia dataset suggests this.

9.2. Countermeasures

9.2.1. Enron with Padding

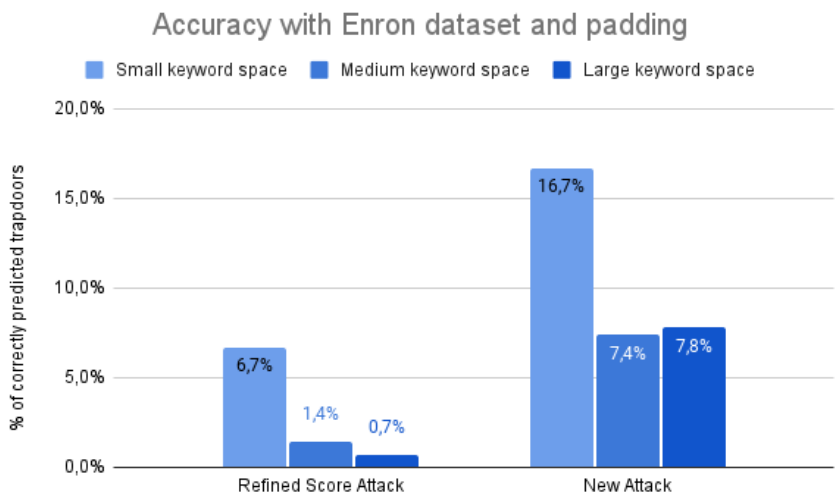


Figure 9.4: Accuracy of the refined score attack, new attack and the subgraph-VL attack with the Enron dataset with the padding countermeasure

In this experiment the padding countermeasure is active. This should make it harder for the attacker to correctly predict trapdoor-keyword pairs. In this attack setting the new attack performs much better than the refined score attack, however the accuracy is still really low, the highest accuracy being 16.7% of the new attack in the smallest keyword universe. The accuracy for the refined score attack is terrible and the accuracy of the new attack is more than double the accuracy of the refined score attack, but the accuracy is still really bad. In this case the new attack is thus more resistant to countermeasures, but still not enough.

9.2.2. Enron with Obfuscation

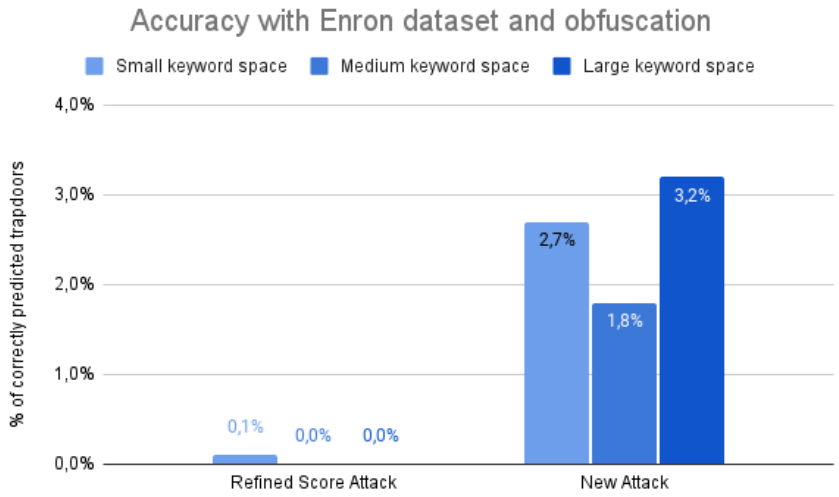


Figure 9.5: Accuracy of the refined score attack, new attack and the subgraph-VL attack with the Enron dataset with the obfuscation countermeasure

In this experiment the obfuscation countermeasure is active. This should make it harder for the attacker to correctly predict trapdoor-keyword pairs. With the obfuscation countermeasure, the situation for both attacks is the same as with the padding countermeasure. The results of the refined score attack are terrible, and the new attack is better, but still very bad. One could say that the new attack is more resistant to countermeasures, but since the highest accuracy is only 3.2%, this is not a good statement to make.

9.2.3. Apache Lucene with Padding

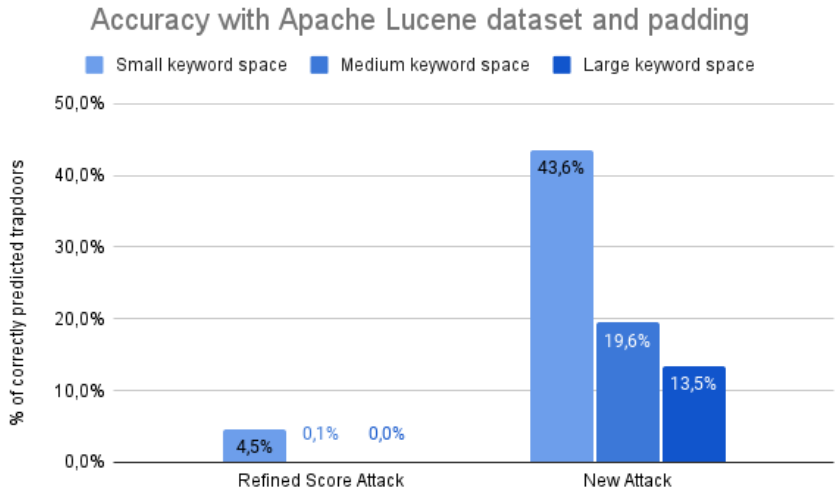


Figure 9.6: Accuracy of the refined score attack, new attack and the subgraph-VL attack with the Apache Lucene with the padding countermeasure

In this experiment the padding countermeasure is active. The difference in accuracy between both of the attacks is very big. The new attack is much better, although for the medium and large keyword spaces the accuracy is still very poor at best. However for the small keyword space the accuracy

is reasonable. The refined score attack scores very bad, even in the small keyword space with an accuracy of 4.5%. The new attack is more resistant against countermeasures. However the padding makes the attack still struggle with correctly predicting the correct trapdoor-keyword pairs.

9.2.4. Apache Lucene with Obfuscation

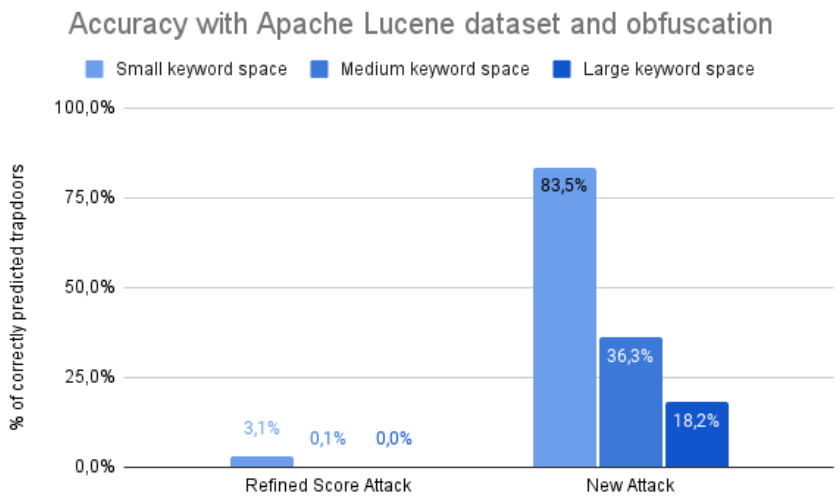


Figure 9.7: Accuracy of the refined score attack, new attack and the subgraph-VL attack with the Apache Lucene dataset with the obfuscation countermeasure

In this experiment the obfuscation countermeasure is active. The difference in accuracy between both of the attacks is now even larger. The new attack is much better in terms of accuracy, although for the medium and large keyword spaces the accuracy is still not very high, but the accuracy is higher than for the padding countermeasure. With the small keyword space the new attack scores exceptionally well, with an accuracy of 83.5%. Compared to now countermeasure, this is a drop of only 14%. The drop in accuracy is much higher with the medium and large keyword space.

9.2.5. Wikipedia with Padding

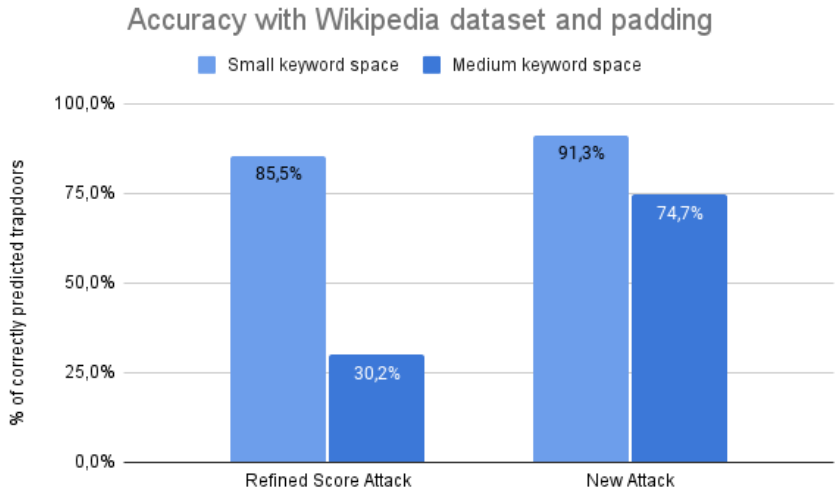


Figure 9.8: Accuracy of the refined score attack, new attack and the subgraph-VL attack with the Wikipedia dataset with the padding countermeasure

In this experiment the padding countermeasure is active. The accuracy of both attacks has dropped a bit with both keyword spaces. However the accuracy of both attacks are still really high and it seems like the countermeasure did not help a lot to lower the accuracy of the attacks. The new attack is more resistant to countermeasures, since its accuracy with the medium keyword space has not dropped a lot compared to the refined score attack. The accuracy of the refined score attack dropped 58.4% with the medium keyword set, whereas the new attack its accuracy has only dropped 15.1%.

9.2.6. Wikipedia with Obfuscation

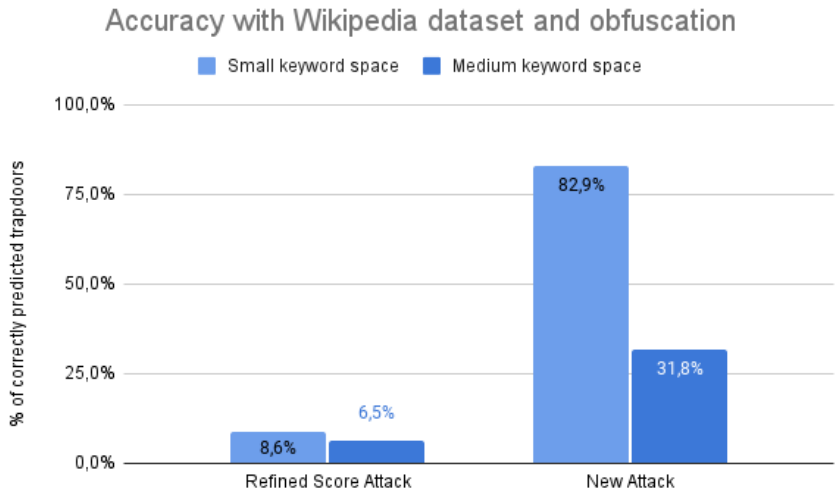


Figure 9.9: Accuracy of the refined score attack, new attack and the subgraph-VL attack with the Wikipedia dataset with the obfuscation countermeasure

In this experiment the obfuscation countermeasure is active. The obfuscation countermeasure has worked very well in reducing the accuracy of the refined score attack. The accuracy of the new attack has also dropped a bit, especially with the medium keyword space. But the accuracy for the small keyword space is still really high. The new attack is more resistant to the obfuscation countermeasure.

9.2.7. Comparison

What can be seen from the results with the countermeasures is that the new attack is not really impacted a lot by them in the small keyword spaces with the Apache Lucene and Wikipedia datasets. The accuracy of the new attack in these cases is still really high. In the medium and larger keyword spaces the accuracy still drops by a lot. But in all cases the new attack performs much better than the refined score attack. This means that the new attack is much more resistant to countermeasures than the refined score attack, however because of the accuracy drops in the medium and large keyword sets, one cannot say that it is resistant to the padding and obfuscation countermeasures.

In the Enron dataset the new attack is also impacted a lot by the countermeasures. This also holds for the smaller keyword universe. The refined score attack performs even worse than the new attack, but what could be said is that in all cases the new attack is more resistant to countermeasures than the refined score attack. But the new attack is not resistant against the countermeasures.

The reason for why the attacks perform much worse with the Enron dataset and not as much with the other two datasets could maybe be found in figure 8.1. In the Enron dataset there is a much smaller keyword universe and the average keyword universe per document is also much smaller in the Enron dataset. Because of this it might be harder for the attacks to predict the right keyword-trapdoor pairs, since there are less distinct keyword-keyword occurrences and keyword occurrences since the keyword universes are smaller. This is just a mere correlation between the table and the results of the attacks, and more research is needed to confirm this correlation, but for now it seems to be a justifiable assumption. This is the same reasoning as for the results with no countermeasures.

10 Conclusion

This thesis shows a new similar-data attack against SSE systems that is highly effective when comparing it to other similar-data attacks and known-data attacks. The accuracy of the new attack is much better, also when there are countermeasures. Now it is time to answer the main research question of this thesis, given in chapter 5. The main research question and sub-questions of this thesis were:

What is the current state of similar-data query recovery attacks and is it possible to create a new and better attack?

- SQ1: What is currently the best similar-data query recovery attack?
- SQ2: How would one improve the best similar-data attack?
- SQ3: What would a similar-data attack together with a file injection attack look like?
- SQ4: Will the new attack be resistant to countermeasures?

As discussed in chapter 4, there are currently at the time of writing this thesis, not many similar-data query recovery attacks on SSE systems. There are only three attacks, and the refined score attack in [9] is the best out of the three in terms of accuracy for the L1 leakage profile. This answers SQ1. The best similar-data query recovery attack is the refined score attack. For more information, please refer back to chapter 4 and chapter 6.

As discussed in chapter 6, the refined score attack has some flaws, even though it is the best similar-data attack. These flaws being the assumption of known keyword-trapdoor pairs by the attacker, not using the keyword occurrence and the RefSpeed parameter. As explained, fixing these flaws could majorly improve the accuracy of the attack, which can be seen from the results in the chapter 9. So this answers SQ2.

In chapter 7, the new attack is shown. This new attack fixes the flaws of the refined score attack discussed in chapter 6, and adds a file injection component to the refined score attack. This remediates the first flaw of the refined score attack. In this particular case the file injection attack is used to create a ”base-mapping” of keyword-trapdoor pairs. With the keyword-keyword occurrences of these pairs being the initial sub-matrix that is used to calculate the scores for the attack. Only a few files need to be injected for the attack to know some of the keyword-trapdoor pairs. After that, the attack can continue with predicting the rest of the queries. So to answer SQ3, refer to chapter 7 in which the file injection component is used to create the first few keyword-trapdoor pairs. The pseudocode of the attack is also given in chapter 7.

Finally to answer SQ4, one can refer to the results in chapter 9. The new attack is not resistant to countermeasures, but it performs much better than the refined score attack, and even though the accuracy is not very high in all cases, the results are still very usable, and in some cases still very high (>80%) for the Apache Lucene and Wikipedia dataset with a small keyword universe. The new attack is more resistant to countermeasures in the case of a larger keyword universe. However the new attack is not completely resistant against countermeasures, since the accuracy still drops quite a lot when comparing the accuracy with the countermeasures, with the accuracy of no countermeasures.

Now that the sub-questions have been answered, the main research question can be answered. Like already said many times before there is not a lot of research into similar-data attacks. Most research focuses on known-data attacks, since it is somewhat easier to get better results when more known-data is given to the attacks. However with this research another attack has been created that is better than most known-data attacks without any assumptions of known-data or known keyword-trapdoor pairs, the only assumption being that the attacker is able to insert files into the SSE systems and that the attacker has a similar dataset. Obviously, as explained before in chapter 4, the similarity of the real dataset and similar dataset also determines the accuracy of the attack, and the less similar the datasets are, the lower the accuracy of a similar-data query recovery attack will be. Nonetheless this attack is still a step in the right direction, and with not much research being done on similar-data attacks, the future is still bright for even better attacks than the one described in this paper.

To answer the second part of the main research question, yes it is possible to create a new and better similar-data query recovery attack. As shown in the previous chapter, the new attack is currently (at the time of writing, to the best of my knowledge), the best similar-data attack out there. Albeit that it has some file injection components.

References

[1] Michel Abdalla et al. “Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions”. In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 205–222. ISBN: 978-3-540-31870-5.

[2] Alexandre Anzala-Yamajako et al. “No Such Thing as a Small Leak: Leakage-Abuse Attacks Against Symmetric Searchable Encryption”. In: *Communications in Computer and Information Science*. E-Business and Telecommunications 14th International Joint Conference, ICETE 2017, Madrid, Spain, July 24-26, 2017, Revised Selected Paper 990 (Jan. 2019), pp. 253–277. URL: https://hal.archives-ouvertes.fr/hal-01990354.

[3] Laura Blackstone, Seny Kamara, and Tarik Moataz. *Revisiting Leakage Abuse Attacks*. Cryptology ePrint Archive, Paper 2019/1175. https://eprint.iacr.org/2019/1175. 2019. URL: https://eprint.iacr.org/2019/1175.

[4] Dan Boneh and Brent Waters. “Conjunctive, Subset, and Range Queries on Encrypted Data”. In: *Theory of Cryptography*. Ed. by Salil P. Vadhan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 535–554. ISBN: 978-3-540-70936-7.

[5] Dan Boneh et al. “Public Key Encryption with Keyword Search”. In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by Christian Cachin and Jan L. Camenisch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 506–522. ISBN: 978-3-540-24676-3.

[6] David Cash et al. “Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation”. In: Jan. 2014. ISBN: 1-891562-35-5. DOI: 10.14722/ndss.2014.23264.

[7] Melissa Chase and Seny Kamara. “Structured Encryption and Controlled Disclosure”. In: *Advances in Cryptology - ASIACRYPT 2010*. Ed. by Masayuki Abe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 577–594. ISBN: 978-3-642-17373-8.

[8] Guoxing Chen et al. “Differentially Private Access Patterns for Searchable Symmetric Encryption”. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 810–818. DOI: 10.1109/INFOCOM.2018.8486381.

[9] Marc Damie, Florian Hahn, and Andreas Peter. “A Highly Accurate Query-Recovery Attack against Searchable Encryption using Non-Indexed Documents”. In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 143–160. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/damie.

[10] *Enron Email Dataset*. URL: https://www.cs.cmu.edu/~enron/.

[11] Oded Goldreich and Rafail Ostrovsky. “Software Protection and Simulation on Oblivious RAMs”. In: *Journal of the ACM* 43 (Jan. 1996). DOI: 10.1145/233551.233553.

[12] *Google Trends*. URL: https://trends.google.com/.

[13] Warren He et al. “ShadowCrypt: Encrypted Web Applications for Everyone”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’14. Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 1028–1039. ISBN: 9781450329576. DOI: 10.1145/2660267.2660326. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/2660267.2660326.

[14] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. “Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation.” In: *NDSS*. The Internet Society, 2012. URL: http://dblp.uni-trier.de/db/conf/ndss/ndss2012.html#IslamKK12.

[15] Wang Jie, Xiao Yu, and Ming Zhao. “A Novel Dynamic Ranked Fuzzy Keyword Search Over Cloud Encrypted Data”. In: Aug. 2014. DOI: 10.1109/DASC.2014.25.

[16] Seny Kamara and Tarik Moataz. “Computationally Volume-Hiding Structured Encryption”. In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Yuval Ishai and Vincent Rijmen. Cham: Springer International Publishing, 2019, pp. 183–213. ISBN: 978-3-030-17656-3.

[17] Seny Kamara and Charalampos Papamanthou. “Parallel and Dynamic Searchable Symmetric Encryption”. In: *Financial Cryptography and Data Security*. Ed. by Ahmad-Reza Sadeghi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 258–274. ISBN: 978-3-642-39884-1.

[18] Sean Michael Kerner. *CipherCloud debuts searchable strong encryption for the cloud*. Oct. 2013. URL: https://www.eweek.com/security/ciphercloud-debuts-searchable-strong-encryption-for-the-cloud/.

[19] Dalia Khader. “Public Key Encryption with Keyword Search Based on K-Resilient IBE”. In: *Computational Science and Its Applications - ICCSA 2006*. Ed. by Marina Gavrilova et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 298–308. ISBN: 978-3-540-34076-8.

[20] Billy Lau et al. “Mimesis Aegis: A Mimicry Privacy Shield–A System’s Approach to Data Privacy on Public Cloud”. In: *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 33–48. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/lau.

[21] Chang Liu et al. “Search pattern leakage in searchable encryption: Attacks and new construction”. In: *Information Sciences* 265 (2014), pp. 176–188. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2013.11.021. URL: https://www.sciencedirect.com/science/article/pii/S0020025513008293.

[22] Zheli Liu et al. *FFSSE: Flexible Forward Secure Searchable Encryption with Efficient Performance*. Cryptology ePrint Archive, Paper 2017/1105. https://eprint.iacr.org/2017/1105. 2017. URL: https://eprint.iacr.org/2017/1105.

[23] Edward Loper and Steven Bird. “NLTK: The Natural Language Toolkit”. In: *CoRR* cs.CL/0205028 (2002). URL: http://dblp.uni-trier.de/db/journals/corr/corr0205.html#cs-CL-0205028.

[24] *Lucene™ mailing lists and irc*. URL: https://lucene.apache.org/core/discussion.html.

[25] Qiumao Ma et al. “SE-ORAM: A Storage-Efficient Oblivious RAM for Privacy-Preserving Access to Cloud Storage”. In: *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*. 2016, pp. 20–25. DOI: 10.1109/CSCloud.2016.24.

[26] Simon Oya and Florian Kerschbaum. “Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption”. In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 127–142. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/oya.

[27] Sarvar Patel et al. *Mitigating Leakage in Secure Cloud-Hosted Data Structures: Volume-Hiding for Multi-Maps via Hashing*. Cryptology ePrint Archive, Paper 2019/1292. https://eprint.iacr.org/2019/1292. 2019. URL: https://eprint.iacr.org/2019/1292.

[28] Jason Perry et al. “Leakage-Abuse Attacks Against Searchable Encryption”. In: Oct. 2015. DOI: 10.1145/2810103.2813700.

[29] Rishabh Poddar et al. “Practical Volume-Based Attacks on Encrypted Databases”. In: *2020 IEEE European Symposium on Security and Privacy (EuroSP)*. 2020, pp. 354–369. DOI: 10.1109/EuroSP48549.2020.00030.

[30] David Pouliot and Charles V. Wright. Oct. 2016. URL: http://library.usc.edu.ph/ACM/SIGSAC%5C%202017/ccs/p1341.pdf.

[31] Help Net Security. *Bitglass granted patent on Searchable Cloud Encryption*. July 2015. URL: https://www.helpnetsecurity.com/2015/07/09/bitglass-granted-patent-on-searchable-cloud-encryption/.

[32] Zhiwei Shang et al. “Obfuscated Access and Search Patterns in Searchable Encryption”. In: Jan. 2021. DOI: 10.14722/ndss.2021.23041.

[33] David Shapiro and James Morris. *PlainTextWikipedia*. URL: https://github.com/daveshap/PlainTextWikipedia.

[34] *Simple English Wikipedia*. URL: https://simple.wikipedia.org/wiki/Simple_English_Wikipedia.

[35] Branka Vuleta. *How much data is created every day? +27 staggering stats*. Oct. 2021. URL: https://seedscientific.com/how-much-data-is-created-every-day.

[36] Peter Willett. "The Porter stemming algorithm: Then and now". In: *Program electronic library and information systems* 40 (July 2006). DOI: 10.1108/00330330610681295.

[37] Lei Xu et al. "Hardening Database Padding for Searchable Encryption". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 2019, pp. 2503–2511. DOI: 10.1109/INFOCOM.2019.8737588.

[38] Lei Xu et al. "Interpreting and Mitigating Leakage-Abuse Attacks in Searchable Symmetric Encryption". In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 5310–5325. DOI: 10.1109/TIFS.2021.3128823.

[39] Rui Zhang and Hideki Imai. "Combining Public Key Encryption with Keyword Search and Public Key Encryption". In: *IEICE Transactions* 92-D (May 2009), pp. 888–896. DOI: 10.1587/transinf.E92.D.888.

[40] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. "All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption". In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 707–720. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/zhang.

[41] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. "VABKS: Verifiable attribute-based keyword search over outsourced encrypted data". In: *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 2014, pp. 522–530. DOI: 10.1109/INFOCOM.2014.6847976.

A

Raw email from Enron dataset

The raw email of user allen-p in the _sent_mail folder number 13:

```
Message-ID: <32300323.1075855378519.JavaMail.evans@thyme>
Date: Wed, 2 May 2001 12:36:00 -0700 (PDT)
From: phillip.allen@enron.com
To: james.steffes@enron.com
Subject:
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-From: Phillip K Allen
X-To: James D Steffes <James D Steffes/NA/Enron@Enron>
X-cc:
X-bcc:
X-Folder: \Phillip_Allen_Jan2002_1\Allen, Phillip K.\'Sent Mail
X-Origin: Allen-P
X-FileName: pallen (Non-Privileged).pst

Jim,
Is there going to be a conference call or some type of weekly meeting about all the
regulatory issues facing California this week?  Can you make sure the gas desk is included.
Phillip
```

B

Raw email from the Apache dataset

The raw email from the mailing list of December 2009:

```
    Return-Path:
<java-user-return-44049-apmail-lucene-java-user-archive=lucene.apache.org@lucene.apache.org>
Delivered-To: apmail-lucene-java-user-archive@www.apache.org
Received: (qmail 90801 invoked from network); 11 Dec 2009 17:21:29 -0000
Received: from hermes.apache.org (HELO mail.apache.org) (140.211.11.3)
by minotaur.apache.org with SMTP; 11 Dec 2009 17:21:29 -0000
Received: (qmail 4783 invoked by uid 500); 11 Dec 2009 17:21:27 -0000
Delivered-To: apmail-lucene-java-user-archive@lucene.apache.org
Received: (qmail 4682 invoked by uid 500); 11 Dec 2009 17:21:27 -0000
Mailing-List: contact java-user-help@lucene.apache.org; run by ezmlm
Precedence: bulk
List-Help: <mailto:java-user-help@lucene.apache.org>
List-Unsubscribe: <mailto:java-user-unsubscribe@lucene.apache.org>
List-Post: <mailto:java-user@lucene.apache.org>
List-Id: <java-user.lucene.apache.org>
Reply-To: java-user@lucene.apache.org
Delivered-To: mailing list java-user@lucene.apache.org
Received: (qmail 4671 invoked by uid 99); 11 Dec 2009 17:21:27 -0000
Received: from nike.apache.org (HELO nike.apache.org) (192.87.106.230)
by apache.org (qpsmtpd/0.29) with ESMTP; Fri, 11 Dec 2009 17:21:27 +0000
X-ASF-Spam-Status: No, hits=-0.0 required=10.0
tests=SPF_HELO_PASS,SPF_PASS
X-Spam-Check-By: apache.org
Received-SPF: pass (nike.apache.org: domain of rstaveley@seseit.com designates
209.97.205.201 as permitted sender)
Received: from [209.97.205.201] (HELO mail.seseit.com) (209.97.205.201)
by apache.org (qpsmtpd/0.29) with ESMTP; Fri, 11 Dec 2009 17:21:18 +0000
Received: by mail.seseit.com (Postfix, from userid 65534)
id 67D4F183C6; Fri, 11 Dec 2009 17:20:50 +0000 (GMT)
X-Spam-Checker-Version: SpamAssassin 3.1.7-deb3 (2006-10-05) on
mini.seseit.net
X-Spam-Level:
Received: from Bigun (bimport.seseit.net [78.143.212.168])
by mail.seseit.com (Postfix) with ESMTP id D9480183C6
for <java-user@lucene.apache.org>; Fri, 11 Dec 2009 17:20:43 +0000 (GMT)
From: "Rob Staveley \(Tom\)" <rstaveley@seseit.com>
To: <java-user@lucene.apache.org>
Subject: Lucene 3.0.0 writer with a Lucene 2.3.1 index
```

Date: Fri, 11 Dec 2009 17:20:56 -0000
Message-ID: <07c601ca7a86$4de99340$e9bcb9c0$@com>
MIME-Version: 1.0
Content-Type: text/plain;
charset="us-ascii"
Content-Transfer-Encoding: 7bit
X-Mailer: Microsoft Office Outlook 12.0
Thread-Index: Acp6hkvFrgUvDtBcTve1/rQYNBc1ZA==
Content-Language: en-gb
X-seseit-ltd-MailScanner: Found to be clean, Found to be clean
X-seseit-ltd-MailScanner-From: rstaveley@seseit.com
X-Virus-Checked: Checked by ClamAV on apache.org
X-Old-Spam-Status: No, score=0.0 required=5.0 tests=none autolearn=failed
version=3.1.7-deb3, No

I'm upgrading from 2.3.1 to 3.0.0. I have 3.0.0 index readers ready to go into production and writers in the process of upgrading to 3.0.0.

I think understand the implications of http://wiki.apache.org/lucene-java/BackwardsCompatibility#File_Formats for the upgrade, but I'd love it if someone could validate my following assumptions.

1. My 2.3.1 indexes have compressed fields in them, which the 3.0.0 readers work nicely with, as expected. I should assume that my 3.0.0 readers will continue to handle 2.3.1 indexes OK.

2. Presumably Lucene all future 3.x index readers will continue to handle compressed fields and we should only anticipate Lucene 4.x choking on them.

I was naively expecting my index directories to grow when my 3.0.0 index writer merged the 2.3.1 indexes and/or optimize()'d them converting them to 3.0.0. However, I don't see that. Presumably that means that....

3. Documents added to existing 2.3.1 indexes will be added conforming to 3.0.0, but existing documents in the index will continue to have compressed content and old documents can coexist happily with the new ones, and my indexes will become a mixture of 2.3.1 and 3.0.0.

4. I should use http://lucene.apache.org/java/2_9_1/api/all/org/apache/lucene/util/Version.html#LUCENE_23 for the StandardAnalyzer and QueryParser in mixed indexes in 3.0.0 if I want to handle analysis consistently, or go for LUCENE_CURRENT if I want to handle the new content "better" (bearing in mind that the new content will eventually replace the old content anyhow).

5. I should use http://lucene.apache.org/java/3_0_0/api/all/org/apache/lucene/analysis/StopFilter.html#StopFilter%28boolean,%20org.apache.lucene.analysis.TokenStream,%20java.util.Set%29 with enablePositionIncrements=false in mixed indexes in 3.0.0 if I want to handle analysis consistently, or go for enablePositionIncrements=true if I want to handle the new content "better" (bearing in mind that the new content will eventually replace the old content anyhow).

To unsubscribe, e-mail: java-user-unsubscribe@lucene.apache.org
For additional commands, e-mail: java-user-help@lucene.apache.org