# Inject Less, Recover More

## Unlocking the Potential of Document Recovery in Injection Attacks against SSE

Manning Zhang (5140544)

# Inject Less, Recover More

## Unlocking the Potential of Document Recovery in Injection Attacks against SSE

by

Manning Zhang (5140544)

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday August 30, 2023 at 12:00 PM.

| | | |
|---|---|---|
| Student number: | 5140544 | |
| Project duration: | November, 2022 – August, 2023 | |
| Thesis committee: | Prof. George Smaragdakis | TU Delft, thesis Advisor |
| | Dr. Kaitai Liang | TU Delft, daily supervisor |
| | Dr. Jérémie Decouchant | TU Delft, committee member |

Cover: Magnifying glass by Peggy_Marco (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

*Completing this master's thesis signifies the fulfillment of my journey as a master's student at TU Delft, an experience that has been both challenging and rewarding. It also marks a significant milestone for me to reflect on my personal growth and development during this academic endeavor.*

*I am glad and proud to present my work 'Unlocking the Potential of Document Recovery in Injection Attacks against SSE'. This research would not have been possible without the guidance and support of numerous individuals. I extend my heartfelt thanks to my supervisor, Dr. Kaitai Liang, for providing the initial idea and direction for this work. I am also indebted to my co-supervisor, Zeshun Shi, and Ph.D. student, Huanhuan Chen, for their active support and assistance throughout our biweekly meetings. Additionally, I am grateful to my thesis advisor, Prof. Georgios Smaragdakis, for his valuable feedback and insights at every stage evaluation of the project. Lastly, I would like to express my appreciation to Dr. Jérémie Decouchant for his participation in my thesis committee and for dedicating his time to evaluate my work.*

*Manning Zhang (5140544)*
*Delft, August 2023*

# Abstract

*Searchable symmetric encryption (SSE) is an encryption scheme that allows a single user to perform searches over an encrypted dataset. The advent of dynamic SSE has further enhanced this scheme by enabling updates to the encrypted dataset, such as insertions and deletions. In dynamic SSE, attackers have employed file injection attacks, initially proposed by Cash et al. (CCS 2015), to obtain sensitive information. These attacks have shown impressive performance with 100% accuracy and no prior knowledge requirement. However, they fail to recover queries with underlying keywords not present in the injected files. To address these limitations, our research introduces a novel attack strategy that incorporates the idea of inference attacks relying on uniqueness in leakage patterns. The goal is to achieve an amplified effect in query recovery. Additionally, we propose a keyword classification based on their access patterns, which helps identify the current limitation of query recovery in reference attacks. With our proposed attack, we demonstrate a minimum query recovery rate of 1.3 queries per injected keyword with a 10% data leakage of real-life datasets. Furthermore, our findings initiate further research to overcome challenges associated with non-distinctive keywords faced by inference attacks.*

# Contents

# 1

# Introduction

The information era has ushered in an abundance of digital data, posing significant challenges for local storage devices to handle such massive amounts of personal information. In response to this growing demand for efficient data storage solutions, cloud services have emerged as a practical solution. However, storing data in plaintext on third-party servers raises serious concerns about sensitive data leakage and privacy breaches. To address this critical issue, data encryption has become a fundamental requirement, even mandated by General Data Protection Regulation (GDPR), to ensure data security and protect users' privacy.

While encryption effectively safeguards the confidentiality of outsourced data, it comes with certain limitations, particularly in terms of functionality. One significant challenge arises when performing searches over the encrypted content, as each document is encrypted as a unified entity, making selective keyword-based searches impossible.

To address this limitation, Song et al. introduced the concept of Searchable Symmetric Encryption (SSE) in 2000 [36]. SSE is an encryption technology that enables efficient search operations on remotely stored datasets while ensuring the confidentiality of the data. In SSE, each plaintext document is transformed into an encoded stream of keywords, which is then XORed with a stream of keys and uploaded to a server assumed to be honest-but-curious. When a user wants to search for documents containing a specific keyword, they use their private key to encrypt the keyword into a query and send it to the server. The server replies with the encrypted set of documents matching the query, which the user can then decrypt to obtain the plaintext documents.

Since its introduction, numerous scholars in this field have dedicated their efforts to enhancing and fine-tuning searchable encryption (SE) techniques. For example, researchers have explored the feasibility of employing public key searchable encryption, allowing encrypted data to be shared among multiple clients [4, 20]. Moreover, the exploration of dynamic searchable encryption has empowered users with greater control over their encrypted datasets, enabling functionalities such as addition and deletion to be implemented effectively [7, 27].

Since the inception of SSE, ensuring provable security has been a fundamental goal, ensuring that the underlying plaintext of encrypted documents remains protected and accessible only to the private key holder. However, in 2012, Islam et al. [17] raised concerns about potential security flaws in SSE with their work on IKK . Their research initiated the discussion about the possibility of an attacker inferring sensitive information by analyzing leaked documents and access patterns. While it was later proven that IKK is not practical in real-life scenarios due to its high assumption of full data leakage in plaintext, it successfully raised awareness about the importance of considering access patterns in SSE security.

Subsequent research on SSE attacks has been dedicated to exploring the extent to which queries can be recovered by leveraging leakage patterns. Among these attacks, those that assume the existence of data leakage are referred to as inference attacks, such as Count [8], ShadowNemesis [31], Subgraph [3], LEAP [28], and VAL [24]. On the other hand, attacks that do not rely on data leakage are known as file injection attacks [8, 44, 3, 43].

These attacks have distinct strengths that make them suitable for different scenarios. Inference attacks are well-suited for static SSE schemes with data leakage, and file injection attacks are more

advantageous for dynamic SSE schemes, where the attacker aims to achieve a high query recovery rate without relying on data leakage.

Because the success of both inference and file injection attacks hinges on the accuracy of the exploited leakage patterns, consequently, countermeasures have been devised to thwart these types of attacks, and two commonly used approaches are padding and obfuscation [6, 23, 34, 10, 32, 41, 18]. These techniques aim to enhance the security of SSE by introducing noise to obscure the access patterns, making it more challenging for attackers to infer the correct relationship between queries and their corresponding encrypted documents.

As we embark on our research journey, our primary focus is to thoroughly review and analyze existing SSE attacks, comprehending their development trends, strengths, innovative aspects, and limitations. Moreover, we aim to address some of the limitations present in these attacks, striving to explore their potential for improvement. Our intention is not to compromise the security of SSE schemes but rather to discover any potential vulnerabilities that might be exploited. By uncovering these flaws, we can pave the way for future SSE techniques to enhance the overall security level of the schemes.

In our research, we have chosen to focus on making improvements to file injection attacks, specifically aiming to reduce the number of required injected documents. To achieve this goal, we have taken inspiration from inference attacks and incorporated their ideas into the file injection attack approach.

## 1.1. Research Question

The trajectory of our research can be viewed as a journey of answering encountered research questions, and through thorough investigations, we arrived at our final outcome. At the beginning of our research, the main guiding question was:

*What improvements can be made to enhance the efficiency and effectiveness of the state-of-the-art SSE attacks?*

As we delved deeper into the related works, our research questions evolved to:
*What are the untapped directions that can be explored to further enhance SSE attacks?*

To address this question, we proposed various directions that could be explored. Eventually, we chose the direction assuming ”providing data leakage to file injection attacks” and updated the research question to:
*Why is the assumption of data leakage being available to file injection attacks valid?*
*How can we leverage partial data leakage to reduce the size of injected documents?*

These question directly led us to the initial try of proposed attack. However, during the first experiment, we observed unexpected query recovery results, which led to additional sub-questions:
*What are the reasons behind these unexpected results?*
*What factors contribute to the consistent correlation between a higher query recovery rate and a higher frequency of occurrence in the keyword universe?*

By addressing these two questions, we derived our final solution.

## 1.2. Contribution

Our study makes significant contributions in two main areas.

Firstly, we conducted a thorough analysis of the dataset and introduced a novel classification of keywords based on their possession of a unique access pattern. This classification sheds light on the inherent limitations of inference attacks, providing a comprehensive understanding of their effectiveness and challenges.

Secondly, we proposed a new attack framework that combines the strengths of both inference and file injection attacks. Under this framework, we introduced four strategies, with the first three being the initial exploration of introducing an amplification effect in file injection attacks, and the last strategy emerging as the most effective one capable of achieving the amplification effect while also overcoming the inherent limitations of inference attacks.

We evaluated the performance of our proposed attack using three real-life datasets, and by observing the ratio of injected keywords over recovered keywords being larger than 1, we validated the successful achievement of the amplification effect. Additionally, by illustrating the proportion of recovered keywords being greater than the proportion of keywords with no occurrence pattern, we confirmed that our attack surpassed the query accuracy limitations of inference attacks.

Through these contributions, our study provides valuable insights and advancements in the field of SSE attacks, offering novel techniques and a deeper understanding of their capabilities and limitations.

## 1.3. Thesis Structure

The thesis is structured as follows:

**Chapter 2** provides an in-depth explanation of the background knowledge related to Searchable Symmetric Encryption (SSE). This chapter covers the SSE framework, various leakage patterns, and the categories of existing SSE attacks.
**Chapter 3** presents a comprehensive analysis of the SSE attacks that have been studied. The attacks are presented in chronological order, and their novelties, core ideas, experimental results, and vulnerabilities are thoroughly examined. Based on the conclusions drawn from the analysis, intuitive insights for future development are provided.
**Chapter 4** presents the basic idea, process, experimental results, and conclusions of our first proposed attack.
**Chapter 5** explores the problems encountered during the initial design of a new SSE attack and delves into the underlying reasons behind the unsatisfactory results.
**Chapter 6** introduces our second attempt at proposing a new SSE attack. This chapter provides an in-depth explanation of the core idea, detailed process, and countermeasure considerations.
**Chapter 7** presents comprehensive experimental results of our newly proposed attack.
**Chapter 8** serves as the concluding chapter of the thesis. It includes a summary of the work, limitations encountered during the research, and potential future directions for further investigation.

# 2 Background

In this section, we will initially explore the concept of Searchable Symmetric Encryption and examine the underlying factors that contribute to system leakage, subsequently providing advantages for potential SSE attacks.

## 2.1. Searchable Encryption

Searchable Encryption (SE) is a technique that encrypts plaintext files into a searchable form, allowing only authorized users to retrieve specific data containing wish keywords from the encrypted data stored in the cloud. The main objective of SE is to provide a search functionality while preserving the privacy of the plaintext content, ensuring that the cloud server remains oblivious to the actual data. SE schemes typically consist of two stages: $Setup$ and $Search$, involving both users and servers in the process.

During the $Setup$ phase, the user's collection of plaintext data ($D$) is encrypted into encrypted documents ($ED$) in a searchable format, which are then stored on the server. In the $Search$ stage, the user encrypts the search keywords ($W$) into encrypted tokens ($Q$) and sends the queries to the server. The server retrieves the corresponding encrypted documents that match the search criteria and returns them to the user. The user can subsequently decrypt the encrypted documents locally to obtain the plaintext content.

To facilitate the encryption process, an encrypted search algorithm (ESA) is employed, utilizing various cryptographic techniques that offer different tradeoffs between leakage, expressiveness, and efficiency. Here are some examples of these techniques:

**Fully-Homomorphic Encryption (FHE)** [15]: FHE allows users to perform analytical functions directly on encrypted data without decrypting it, enabling privacy-preserving search operations. However, FHE can be computationally intensive and may introduce significant overhead.

**Oblivious RAM (ORAM)** [16]: ORAM techniques offer strong security in searchable encryption schemes by preserving the confidentiality of user queries through the concealment of access patterns. However, the performance overhead of ORAM can be relatively high, which hinders its widespread implementation in real-world scenarios. Additionally, it is important to note that ORAM does not hide the volume pattern of the encrypted data.

**Property-Preserving Encryption (PPE)** [2, 1]: Property-preserving encryption techniques deliberately preserves certain properties of the plaintext data, enabling efficient search and query operations while maintaining privacy. The level of expressiveness and the types of supported operations may vary depending on the specific property-preserving encryption scheme used.

**Functional Encryption (FE)** [4, 12]: FE enables selective access to specific functionalities of encrypted data, granting fine-grained control over access rights and offers a higher level of security compared to PPE. However, FE has certain limitations. Firstly, it does not provide protection for search tokens. Secondly, the efficiency of search operations in FE can be relatively low, as the server needs to attempt

decryption on each ciphertext in the encrypted dataset. This can result in a linear time complexity with respect to the size of the data.

**Searchable/Structured Symmetric Encryption (SSE/STE)** [36, 11, 9]: SSE and STE schemes are specifically designed for efficient and secure searching over encrypted data, striking a balance between leakage, expressiveness, and efficiency. The level of leakage and the supported search functionalities can vary across different SSE/STE schemes, requiring careful consideration and selection.

Based on the specific requirements of ownership and access control, SE can be categorized into two main branches: symmetric and asymmetric. In the symmetric branch, the term 'user' refers to a single entity, which can represent an individual, an organization, or any other authorized party. This user possesses the private key required for encryption and decryption operations.

On the other hand, in the asymmetric branch [4], the term 'user' refers to multiple entities that share public keys. This enables secure communication and access control within a group of authorized users.

In this paper, our focus will be on the symmetric branch of SE.

## 2.2. Searchable Symmetric Encryption

Searchable Symmetric Encryption (SSE) was initially introduced by Song et al. [36] in 2000, marking the beginning of the SE field. Under the umbrella of SSE, there are two sub-branches: static SSE and dynamic SSE.

### 2.2.1. Static SSE

Static SSE refers to a collection of polynomial-time algorithms denoted as $SSE = Enc, QueryGen, Search$. The input for SSE includes a set of documents $D$ intended for storage on a cloud server. Each document $d_i$ can be represented as a set of keywords $W_{d_i}$. The input also includes a private key $k$ that is utilized by the encryption algorithm.

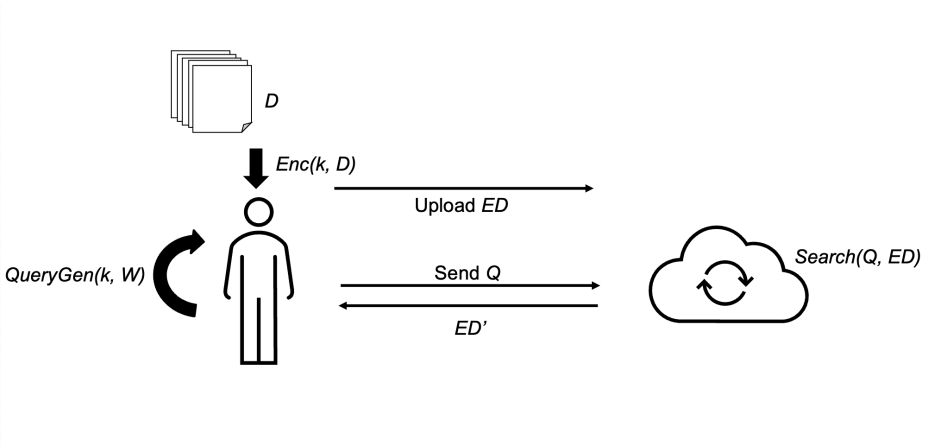



**Figure 2.1:** static SSE

$ED \leftarrow Enc(k, D)$**:** Given a set of plaintext documents $D = \{d_1, ..., d_n\}$ and a private key $k$, the encryption algorithm utilizes an ESA to generate a set of encrypted documents $ED = \{ed_1, ..., ed_n\}$.

$Q \leftarrow QueryGen(k, W)$**:** The query generation algorithm takes a set of specific keywords $W = \{k_1, ..., k_a\}$ that the user intends to search. It produces a set of queries $Q = \{q_1, ..., q_a\}$.

$ED' \leftarrow Search(Q, ED)$**:** The search algorithm utilizes the generated queries $Q$ to perform a search over the encrypted documents $ED$. It returns a subset of encrypted documents $ED' = \{ed_1, ..., ed_b\}$ that contain the desired keywords $W$, where $ED' \in ED$.

### 2.2.2. Dynamic SSE

Dynamic SSE (DSSE) [19, 7, 27], which enables efficient maintenance and dynamic updates of the outsourced dataset, was officially introduced by Kamara et al. [19]. This solution extends the capabilities of existing SSE schemes by incorporating functionalities such as adding and removing data.

DSSE eliminates the need for users to reconstruct all indexes when updating the encrypted contents stored on the server. This alleviates the heavy computational burden, leading to improved performance and enhanced usability of SSE schemes.

During the update process of the dataset in Dynamic Searchable Symmetric Encryption (DSSE), there is a potential vulnerability where the hash value of newly added keywords can be leaked. In response to this, Stefanov et al. [37] emphasized the importance of forward and backward privacy in implementing DSSE.

Forward privacy refers to the server having no knowledge about whether a newly inserted document contains a keyword that was previously searched for. Backward privacy, on the other hand, ensures that queries cannot be searched over deleted documents.

Following the proposal of forward privacy, there has been increased attention towards achieving forward security in Dynamic SSE schemes. The formal definition of forward security was provided by Bost et al. [5] in 2016. In 2017, Kim et al. [21] proposed a dynamic SSE scheme that not only supports efficient updates but also provides forward security, further advancing the research in this field.

## 2.3. Leakage

While encryption in SSE schemes provides a certain level of security by ensuring the confidentiality of the encrypted contents, it is important to acknowledge that SSE schemes are not employed in isolation, which introduces vulnerabilities and potential leakages.

One form of leakage is known as leakage patterns, which involve the disclosure of relationships between the identities of encrypted documents and queries. These leakages can occur during the transmission of messages between users and servers.

Another type of leakage is referred to as data leakage, which involves the unintentional disclosure of plaintext content.

Both leakage patterns and data leakage present potential vulnerabilities in SSE systems, as they can enable adversaries to gather additional information or deduce sensitive details about the encrypted data.

### 2.3.1. Leakage Pattern

The messages exchanged between users and servers can potentially be intercepted by a third party. Intercepted messages consist of encrypted data or keywords, which may not reveal the actual contents but can still provide adversaries with valuable information through the analysis of patterns. Since deterministic encryption is used in SSE, long-term observation enables adversaries to discern patterns between queries and the identities of encrypted documents.

The identity of an encrypted document can be established based on several properties associated with it. These properties include the encrypted identifier of the document, its size, or the number of related queries it is associated with. By examining these properties and identifying patterns, adversaries can make inferences about the characteristics and relationships of the encrypted documents.

There are some mostly used pattern in SSE attacks:

**Access Pattern (AP)** is the primary leakage pattern exploited in SSE attacks, providing insights into the inclusion relationship between identifiers of encrypted documents and a set of queries. In previous works such as [3], the AP is defined as a function. It can be represented as a set of binary column vectors, denoted as $(ED(q_1), ..., ED(q_n))$, where $ED(q_j)$ represents a binary column vector with $m$ entries. Each entry in $ED(q_j)$ indicates whether the corresponding encrypted document contains the query $q_j$, with a value of 1 denoting its presence and a value of 0 indicating its absence. The size of the input query set $Q$ is denoted as $n$, and the size of the input encrypted document set $ED$ is represented by $m$.

Mathematically, the AP can be expressed as:

$$AP = ED \times Q \rightarrow [2^m]^n,\ m, n \in \mathbb{N}$$

Prior to the exploration of other leakage patterns in SSE attacks, Cash et al. [8] proposed a 4-level classification framework for access pattern leakage called Leakage Hierarchy. This classification system categorizes leakages into different levels based on their severity. Lower-level leakages correspond to smaller amounts of leaked information. As the leakage level increases, more information about the pattern can be inferred, thereby reducing the complexity for adversaries to uncover the hidden content, and Table 2.1 presents an overview of this framework:

**Table 2.1:** Leakage Hierarchy

| **Level** | **Knowledge Leakage of $Q$ and $ED$** |
|---|---|
| $Level\ 1$ | appearance of only queried tokens |
| $Level\ 2$ | appearance[1] |
| $Level\ 3$ | appearance, position[2] |
| $Level\ 4$ | appearance, position, amount of occurrence[3] |

[1] 'appearance' provides information on the presence or absence of a specific query within the encrypted document.
[2] 'position' reveals the relative placement of the query within the encrypted document.
[3] 'amount of occurrence' indicates how many times a query appears within an encrypted document.

The access pattern used in subsequent SSE attacks, including our own attack, falls under the category of level 2.

The leakage of ownership between identifiers of encrypted documents and queries extends beyond its literal meaning and yields additional information. One such derived pattern is the search pattern.

**Search Pattern (SP)** reveals information about whether two queries are generated from the same keyword or not. In SSE schemes that use deterministic encryption algorithms, the same keyword will always result in the generation of identical queries. In [3], the search pattern is defined as a function:

$$SP = ED \times Q \rightarrow \{0,1\}^{n \times n}$$

This pattern can be utilized to track the frequency of a specific query being searched.

**Volume Pattern (VP)** reveals an inclusion relationship between encrypted documents and queries by utilizing the volumes of encrypted documents as their identifiers. The volume of a document can be determined by the number of words it contains or its size in bytes, denoted as $|ed_i|$. For query $q_j$, the column vector represents the volumes of encrypted documents that contain the query, denoted as $(|ED|)_{ED \in D(q_j)}$. Its mathematical function is defined as:

$$VP = ED \times Q \rightarrow \ [\mathbb{N}]^n$$

**Total Volume Pattern(TVP)** can be derived from the VP or deduced in the case where an SSE scheme hides the individual volumes of encrypted documents but leaks the volume information of the entire set of corresponding encrypted documents for a query. For each query $q_j$, the corresponding volume is $\sum_{D \in D(q_j)} |ED|$.

$$TVP = ED \times Q \rightarrow \ \mathbb{N}^n$$

**Response Length Pattern (RLP)** provides the adversary an insight into the lengths of the responses returned by the server for different queries. This pattern can be derived from both AP and VP.

$$RLP = ED \times Q \rightarrow \mathbb{N}$$

*Atomic Leakage Pattern (ALP)* refers to the leakage pattern that reveals information about each matching document. It encompasses leakage patterns such as the AP and the VP. The ALP can be seen as a category of leakage patterns mentioned in [3], particularly in the context of Subgraph attacks.

### 2.3.2. Data Leakage

While the use of a private key in SSE can ensure the confidentiality of the encrypted content, it is important to acknowledge that the leakage of plaintext content can still occur. This can happen before or during the setup stage when the user possesses the plaintext data.

Data leakage can be attributed to many reasons and here are 4 of them [13]: misconfiguration issues, Zero-Day vulnerabilities, legacy techniques and tools, and social engineering attacks.

Misconfiguration issues arise from errors in configuring networked data systems, resulting in data exposure and leaks. Automation tools can mitigate the risk, but they also need to be correctly configured.

Zero-Day vulnerabilities refer to unknown software vulnerabilities that can be exploited by attackers without the organization's knowledge.

Improper protection of legacy systems and devices, including the misplacement or theft of desktops and USBs, can lead to data leaks.

Social engineering attacks involve deceiving privileged users into revealing sensitive information, such as login credentials.

Furthermore, data leakage can occur in everyday scenarios, such as mistakenly sending personal information to the wrong recipient in an email system.

It is worth noting that human factors are often responsible for the majority of data leakage incidents, as highlighted by Bruce Schneier in his work [33]. Humans are often the weakest link in security systems and frequently contribute to security failures.

In conclusion, data leakage presents a significant challenge for encryption schemes, primarily due to the involvement of human factors.

## 2.4. Attack

In this paper, the attacks under study are categorized as leakage attacks, as they exploit the leakage inherent in SSE schemes, utilizing either leakage patterns or data leakage. They can be further classified into two main categories based on the prior knowledge possessed by the adversary and how that knowledge is leveraged: inference attacks and file injection attacks.

### 2.4.1. Inference Attacks

In the literature, there is a lack of consensus on the standardized terminology for this specific type of attack. In our work, we use the term 'inference attacks' to categorize these attacks instead of referring to them as 'passive attacks'. This choice is based on the fundamental concept behind these attacks, which involves deducing sensitive information by leveraging both observed leakage patterns and data leakage.

By exploiting the obtained data leakage and leakage patterns, adversaries can construct feature comparison functions to assess the similarities between user data and publicly available databases. Through this process, they can continuously infer the keywords used by the user.

Data leakage used for inference attacks does not necessarily have to be a subset of the target database. It can also originate from a dataset with similar data content and structure. Inference attacks that use a subset of the target dataset as prior knowledge are commonly referred to as known data attacks or leakage abuse attacks. On the other hand, inference attacks that utilize a dataset with similar data are known as similar data attacks.

Given that both data leakage and leakage patterns play a role in inference attacks, and the amount of data leakage often impacts the accuracy of the recovery process, the direction for improving inference

attacks is to reduce the size of prior knowledge while increasing the recovery accuracy.

### 2.4.2. File Injection Attacks

File injection attacks, also known as chosen-data or active attacks, specifically exploit leakage patterns without relying on prior knowledge. These attacks aim to achieve a high level of query recovery accuracy by manipulating users into encrypting and uploading a predefined set of files into the system. Such attacks are typically applicable to DSSE systems that support the insertion of new encrypted documents.

Injection attacks were first introduced by Cash et al. [8], where the core concept revolves around deceiving a user into encrypting and uploading a specific set of files, for instance, each containing a single keyword. Through meticulous observation of the timing and/or payload of these newly inserted files, an attacker can establish the corresponding relationship between the database and the inserted documents. Subsequently, when a query is executed and the server responds with one of the inserted files, the attacker can promptly deduce the underlying keyword associated with that particular query.

However, in this 'single keyword' version of file injection attacks, recovering all queries in the query universe requires injecting a number of files equal to the size of the query universe. This increases the likelihood of detection as a large number of suspicious documents are injected into the system.

The goal of improving file injection attacks is to reduce the number of injected documents while still maintaining a relatively high level of query recovery accuracy.

### 2.4.3. Adversarial Model and Attack Target

**Adversarial model [3]:** Snapshot attacks only require access to the collection of encrypted documents, while persistent attacks require access not only to the encrypted data collection but also to the transcripts of the query operations.

**Attack target [3]:** The attack target can be either the queries or the encrypted documents. For query recovery, the adversaries aim to uncover the underlying keywords associated with the queries. for document recovery, the adversaries' goal is to reveal the plaintext content of the encrypted documents or match the identifiers of the encrypted documents to the identifiers of leaked plaintext documents.

## 2.5. Countermeasures

The primary focus of researchers in SSE attacks extends beyond the design of efficient and high-recovery-rate attacks. The importance of this field lies in the identification and understanding of potential vulnerabilities within SSE schemes. By uncovering these vulnerabilities, researchers can contribute to the development of robust and secure SSE systems.

Data leakage in encryption schemes poses a significant challenge, especially due to human factors. In response, current research on countermeasures for SSE attacks focuses on concealing leakage patterns. The two most commonly used countermeasures are padding and obfuscation [6, 23, 34, 10, 32, 41, 18], which involve returning a bogus set of encrypted documents in response to queries. They are possible to resist both inference attacks and file injection attacks.

Padding involves adding extra documents to the server's response, while obfuscation includes both adding and removing queries for the encrypted documents.

Adding documents is preferred as false positives cause less confusion to users. However, attackers can still gain some knowledge of the correct access pattern as it remains a subset of the bogus access pattern.

On the other hand, obfuscation provides stronger protection by removing correct queries, resulting in true positives. This can cause more confusion for attackers, but users may also experience confusion if the decryption stage of the scheme cannot handle the removed contents properly.

To conceal access patterns, countermeasures involve modifying the server's response by altering the set of identifiers of encrypted documents corresponding to queries. The selection of queries that will have bogus search results can be based on various strategies, such as their frequencies or other criteria. For concealing volume patterns, the sizes of documents in the search results are altered.

It is worth noting that in order to counter SSE attacks that exploit access or volume patterns in dynamic schemes, the use of dynamic versions of padding and obfuscation [32, 41] techniques is

necessary. While static versions of these techniques can effectively hide leakage patterns during the setup stage, they do not conceal the patterns associated with subsequently inserted files.

Another approach to hiding the access pattern is through the use of ORAM techniques in schemes, but the volume pattern can still be revealed.

In addition to addressing leakage patterns, specific countermeasures have been designed to resist dynamic SSE attacks. One approach is to prevent the insertion of malicious documents into the encrypted database. This can be achieved by implementing measures to detect and filter out abnormal or suspicious documents. For example, a threshold-based countermeasure, as mentioned in [44], rejects inserting documents with excessively large volumes.

Another approach is to mitigate the attacker's knowledge of the relationship between injected documents and the encrypted database. One example of such a countermeasure is called Vaccine [25], which consists of two algorithms: Self-injection and Remove.

In the Self-injection stage, when a new document is presented to the encryption scheme for encryption, a second file, known as a self-injected file, is generated simultaneously. This self-injected file has the same number of keywords and document size as the original document. By encrypting both files together, it becomes difficult for an attacker to differentiate between the original document and the injected file, thereby making it challenging to decrypt the correct access pattern.

The self-injected documents are intended to be generated with no semantic text. In the Remove stage, a semantic filter is applied using natural language processing techniques to remove these self-injected documents from the database.

# 3

# Related Work

Chapter 2 provides a concise introduction to the two primary classifications of SSE attacks, namely inference attacks and file injection attacks. In this chapter, our focus shifts towards exploring the related works within these two classifications, presented in chronological order to create a cohesive timeline. We will discuss the underlying ideas and novelties behind each of the studied attacks.

By examining the progression of research and advancements in these areas, we aim to gain deeper insights into the existing body of knowledge surrounding SSE attacks.

## 3.1. Inference Attack

A clear pattern emerges when we place all the studied inference attacks on a timeline, revealing that these attacks have made significant progress in achieving satisfactory recovery accuracy while reducing the required data leakage. The required data leakage from the entire dataset [17] to only 10% of it [24].

### 3.1.1. IKK

IKK [17], the first SSE attack proposed by Islam et al. in 2012, played a crucial role in highlighting the significance of access pattern leakage. Before IKK, access pattern leakage was not widely acknowledged as a critical vulnerability that could result in the significant exposure of sensitive information in a dataset. However, the introduction of IKK emphasized the importance of concealing access patterns to enhance the security level of SSE systems.

In the IKK attack, a co-occurrence matrix is employed as a unique 'fingerprint' for each keyword:

**Background knowledge matrix $M$:** The attacker utilizes the data leakage to create a co-occurrence matrix $M$ that captures the probability of two keywords $w_i$ and $w_j$ appearing together in a random plaintext document $d$. Each entry $M_{i,j}$ in the matrix represents the likelihood of the co-occurrence of the specific pair of keywords. Mathematically, $M$ is defined as $M_{i,j} = Pr[(w_i \in d) \wedge (w_j \in d)]$.

The access pattern allows for the calculation of the observed probability of any two queries appearing in the same encrypted document. By employing an optimization algorithm called Simulated Annealing [22], a query can be matched with a keyword based on the closed known probabilities in the co-occurrence matrix $M$.

The IKK experiment was conducted using a real-life dataset called Enron [26]. In their experimental setup, the authors selected the $x$ most frequent keywords as the keyword universe. By setting $x$ to 1500 and the size of the query set to 150, they were able to consistently achieve a query accuracy of 80%. This accuracy was maintained across various sizes of known queries [1].

However, the accuracy of the recovery results is heavily dependent on the accuracy of the background knowledge matrix, which, in turn, relies on the quantity and accuracy of the leaked plaintext

[1] Known query: the set of queries for which the underlying keywords were already known to the attacker.

data. If there is insufficient knowledge about the plaintext leakage or if noise is intentionally added to the matrix as a countermeasure, the overall accuracy of the attack will be greatly reduced.

Additionally, the runtime of the optimization algorithm used in IKK can be quite long, especially when dealing with larger data volumes. This can limit the scalability and practicality of the attack in real-world scenarios.

### 3.1.2. Count

The Count attack [8], developed subsequent to IKK, builds upon the utilization of co-occurrence matrices to distinguish keywords. Its novelty lies in the incorporation of the response length pattern.

Unlike the previous approach, the Count attack does not solely rely on the co-occurrence probabilities of a keyword with other keywords to define its fingerprint. In addition to the co-occurrence information, the Count attack also takes into account the response length associated with the keyword.

If a pair of a keyword and a query have the same unique response length, the attacker immediately considers them a match. These matched pairs are then labeled as the set of known queries.

For keywords that do not have a unique response length, they are considered potential candidates for the corresponding query with the same response length. To determine the most likely match for each query, the candidates undergo a loop of comparison process. The candidate(s) with the closest co-occurrence probabilities are selected as the match(es) for the query. If there is only one candidate remaining, it is considered the final match and joins the known queries.

The Count attack experiment was conducted on two real datasets: the Enron dataset, which was also used in the IKK attack, and an Apache email listing dataset. The Count attack, similar to the IKK attack, utilized the most frequent $x$ keywords as the keyword universe, with $x$ varying from 500 to 6500. In their experiments, they achieved a remarkable recovery accuracy of nearly 100%, outperforming the IKK attack even in scenarios where no prior knowledge of queries was available to Count.

In addition to achieving a stable and outstanding query recovery accuracy, the Count attack has the advantage of a more favorable runtime, as it eliminates the need for an optimization algorithm. However, it still inherits certain vulnerabilities from the IKK attack. These vulnerabilities include the requirement of a sizeable leakage of plaintext data to construct accurate co-occurrence matrices and determine the response lengths of keywords. Moreover, the effectiveness of access pattern padding as a countermeasure remains a concern.

### 3.1.3. Shadow Nemesis

Shadow Nemesis [31], still inspired by the co-occurrence matrix concept, is a notable SSE attack since it was the first attack to demonstrate the adverse impact of data leakage from a similar dataset on other datasets in SSE attacks.

In this attack, the leaked dataset is assumed to be similar to the target dataset. To match these two datasets, Pouliot et al. conducted the matching problem into an existing problem known as Weighted Graph Matching, which is known to be NP-hard. Instead of using the frequency of a single keyword as the comparison criterion, they utilized the frequency of co-occurrence between two keywords. This transformation enabled the utilization of two proposed algorithms for solving the matching problem. The first algorithm is Umeyama [39], which was published in 1988. The second algorithm is the PATH algorithm [42].

The experiment of this attack utilizes two datasets: Enron and Ubuntu [40]. The attack aims to recover the top frequent $n$ keywords, where $n$ ranges from 100 to 1000 with an interval of 100. The recovery results vary depending on the combination of settings used. For example, using PATH and Umeyama over the Enron dataset yields significantly different outcomes. However, overall, the attack's performance tends to decrease as the number of targeted keywords increases.

A significant limitation of this attack is the level of similarity between the leaked dataset and the target dataset, which greatly impacts the results of the inference attack. The difficulty lies in finding a perfect match between the leaked dataset and the target dataset, which naturally poses a challenge to the attack.

### 3.1.4. VA and SVA attacks

The publication of 'Revisiting Leaking Abuse Methods' [3] in 2020 by Blackstone et al. brought attention to the volume leakage pattern in SSE attacks. This work introduced novel inference attack methods, namely Volume Analysis (VA), Selective Volume Analysis (SVA), and Subgraph attacks. VA and SVA specifically leverage only volume-related leakage patterns as an alternative to co-occurrence matrices derived from access patterns. The core idea of these attacks is to identify the correct match by comparing the volumes of files.

In addition, Blackstone et al. aimed to address the assumption of full data leakage in their work. They observed that previous attacks like IKK and Count required access to the entire dataset to achieve their desired results. However, In contrast, the focus of their work in [3] was to explore the possibility of performing inference attacks with partial data leakage. This investigation aimed to make the inference attacks more practical and applicable in real-world scenarios where complete data leakage may not be readily available.

Volume analysis attack (VA) is briefly supported by the idea of assuming that two documents with the same or closest total volume are the same. However, this approach may lack precision when attempting to make matches within a majority of keywords with similar total volumes.

SVA refines the idea of VA by defining a query based on both its corresponding total volume $v_i$ and the response length $l_i$. Both the total volume pattern and response length pattern can be derived from the volume pattern. In the SVA attack, a window is drawn around the query, encompassing all keywords as candidates that have a total volume in the range of $[\delta \cdot v_i, v_i]$, where $\delta < 1$ represents the known rate of data leakage. The next step in SVA is called sensitivity filtering, where candidates are further filtered out if their response length is too far from the expected length. During this process, an error parameter should be determined to allow for some tolerance in response length matching.

The experiments conducted in [3] for inference attacks utilized the same experimental setup, indicating that the Subgraph attack follows a similar setting as described here.

The used keywords were divided into three distinct groups: high-frequency, low-frequency, and relative low-frequency. Unlike previous approaches that solely focused on the most frequent keywords, this categorization allowed for a more comprehensive analysis of different keyword frequencies and their impact on the attacks. The study revealed that the frequency of the selected keywords does indeed have an influence on the final results.

Under the setting of the high-frequency keyword universe, a vertical comparison between the new proposed attacks and existing attacks (IKK and Count), and a horizontal comparison between VA and SVA reveal the following:

Vertical comparison: Assuming full data leakage, both VA and SVA attacks achieved a query recovery rate of approximately 95%. This performance surpassed the 80% recovery rate of the IKK attack and was slightly lower than the 100% recovery rate of the Count attack.

Horizontal comparison: In the situation of partial data leakage, the query recovery accuracy of SVA remained at 0% until the data leakage exceeded 60%, whereas this number was 75% for VA. This indicates that the volume pattern is more valuable than the total volume pattern for inference attacks. However, it is worth noting that even though VA and SVA do not strictly require full data leakage, a sizeable amount of leakage is still needed to achieve meaningful results with both attacks.

However, when a low-frequency keyword universe was used, both VA and SVA shared a similarly poor performance, as none of the attacks achieved a query recovery rate exceeding 20%. This suggests that the effectiveness of the attacks is greatly influenced by the frequency of the selected keywords.

### 3.1.5. Subgraph Attack

The Subgraph [3] attack has a broader applicability in terms of the type of leakage pattern it can exploit. It can be implemented with any atomic leakage patterns, as introduced in Chapter 2, whether it is an access pattern or a volume pattern. This flexibility allows the Subgraph attack to adapt and utilize different types of leakage patterns depending on the specific scenario and dataset.

Simultaneously, the Subgraph attack has taken a significant step closer to achieving satisfactory results with the utilization of partial data leakage, and 'Correct-matching keywords have leaked information that is a subset of the query information' is the answer given by Blackstone et al. This is due to the leaked information of keywords being derived from partially leaked files, while the access information of the query is obtained from the same but complete database.

If the response documents of a keyword form a subset of the response documents of the query, it implies that the identifiers or volumes of the corresponding documents of the keyword should all be included in the query's response, and the response length of the keyword must be greater than $(data\ leakage\ rate \times response\ length\ of\ the\ query) - error$.

Similar to the last step in the Count attack, if there is only one keyword left in the candidate set of a query, it is concluded that this keyword is a match for the query. Subsequently, the matched keyword is removed from the acceptable option sets of other queries. This iterative elimination process progressively narrows down the possible options

Under the high-frequency keyword universe setting, the Subgraph attack shows promising results in query recovery accuracy. With a data leakage rate of 30%, the attack achieves a minimum query recovery accuracy of 50%. Surprisingly, as the data leakage rate decreases to 10%, the accuracy improves to over 60%. Remarkably, when the data leakage rate exceeds 50%, the query recovery rate surpasses 80%.

However, in the low-frequency keyword universe setting, the Subgraph attack's performance is poor. It achieves a maximum query recovery accuracy of 20% when there is full data leakage, and this accuracy drops to 0% when the rate of data leakage is below 10%.

During the reproduction, the refinement step in the Subgraph attack has limitations in effectively differentiating between a small number of files that contain multiple keywords. This is particularly true in the volume version, where it can result in a failure of the attack with extremely low accuracy, even reaching zero. Additionally, when attempting to recreate the attack described in the paper, there is a need to consider the choice of error parameter as the paper mentions this value was set experimentally.

### 3.1.6. Passive attack with weaker assumption

Ning et al. [29, 28] have contributed significantly to the field of inference attacks by proposing two attacks with a similar core idea. Their work has advanced the investigation of inference attacks, bringing it closer to the goal of achieving a higher recovery rate even with partial data leakage. In this subsection, we will provide a summary of the first attack proposed in the paper 'Passive Attacks Against Searchable Encryption' [29], published in 2018.

The core idea of this attack is to leverage the knowledge of matched identifiers between the leaked plaintext documents and the corresponding encrypted documents. With this information, the attacker can then use it to infer matches between keywords and queries based on their occurrence patterns, as they appear in the same set of documents in both the leaked plaintext and encrypted datasets.

However, a significant challenge in this 'first document recovery, then query recovery' approach lies in the strong assumption that the attacker already possesses the knowledge of matched identifiers.

Under the more realistic scenario where attackers have no prior knowledge of the matched identifiers and only have partial data leakage along with corresponding keywords, Ning et al. presented a solution for document recovery. They achieve this by finding matches between the leaked plaintext documents and encrypted documents that have the same unique number of keywords. After that, keywords are matched with queries that appeared in the same set of matched documents.

The experiment conducted in the Enron dataset uses the top 5000 most frequent keywords as the keyword universe. With a data leakage rate of 10%, the query recovery rate reaches around 60%, which is similar to the Subgraph attack's performance. As the data leakage rate increases to 20%, the query recovery rate further improves to 70%, surpassing the Subgraph attack's performance. However, the highest query recovery rate achieved in this experiment is 75% with a data leakage rate of 80%.

In summary, the overall performance of this attack is comparable to that of the Subgraph attack. As an inference attack based on the access pattern, padding is proposed to be an efficient countermeasure to this attack.

### 3.1.7. LEAP

LEAP [28] attack is also proposed by Ning et al. in 2021. After a period of deeper investigation, they reached a higher query accuracy based on the idea of 'first document recovery, the query recovery'.

In LEAP, the uniqueness of documents and keywords is defined by three factors: the unique size of corresponding keywords/documents, the probability of co-occurrence between documents, and their unique appearance.

In the first phase of LEAP, attackers find an initial set of matched leaked and encrypted documents based on their unique counts of keywords and queries, which is similar to the approach used in the work of Ning et al. [29]. However, what sets LEAP apart is the re-introduction of the concept of co-occurrence matrices. This time, the matrices contain the number of keywords shared by two documents, allowing for a more precise and refined matching process.

Based on the matched documents, LEAP can establish links between keywords and queries that have the same occurrence pattern. These matched keywords enable LEAP to discover additional document matches, creating a positive feedback loop that enhances the accuracy of the attack.

LEAP adopts the same keyword universe setting as previous inference attacks, selecting the top 5000 most frequent keywords from the Enron dataset. The document recovery rate remains consistent in the range of 90% to 93% as the data leakage varies from 0.5% to 100%.

The query recovery rate achieved by LEAP ranges from 99.46% to 11.54%, with the percentage of leaked documents varying from 100% to 0.1%. Notably, the query recovery rate surpasses 50% from a leakage rate of 1% of the data, and it exceeds 90% since a leakage rate of 5%.

Despite the effectiveness of access-hiding countermeasures, such as padding, against LEAP, it is still considered the state-of-the-art inference attack with the most satisfactory outcomes, especially when only partial data is leaked.

The utilization of co-occurrence of documents is particularly noteworthy due to its ability to maintain accuracy in determining the number of keywords shared by any two leaked documents, regardless of whether the leakage is partial or complete. This is in contrast to the keyword version of the co-occurrence matrix.

### 3.1.8. VAL

VAL [24] builds upon the foundation established by LEAP and incorporates the use of volume patterns. It is also the first inference attack we have studied that leverages both access and volume patterns.

During the document recovery phase, the attack still identifies potential document matches by considering the unique number of keywords contained in the documents and the distinct occurrence pattern observed over the matched keywords. What sets VAL apart from LEAP is the additional consideration of matching documents based on their unique volume.

The experiment involved three datasets: Enron, Lucene, and Wikipedia. The document recovery rates for all datasets remain stable from a data leakage rate of 0.5% and above. Among them, Lucene achieves the highest document recovery rate, reaching approximately 98.5%, followed by Enron at around 98%, and Wikipedia with a recovery rate of approximately 89.5%.

In terms of query recovery, the rates for all three datasets approach close to 100% at a data leakage rate of 10%.

By incorporating both access pattern and volume pattern, the attack enhances the distinctiveness of documents, resulting in higher accuracy in document recovery. Moreover, relying solely on access pattern-hiding countermeasures is no longer sufficient to resist the attack. To effectively defend against the VAL attack, it is necessary to employ a combination of countermeasures that hide both access and volume patterns.

## 3.2. File Injection Attack

File injection attacks also identify queries to the underlying keywords based on the knowledge of known plaintext. However, unlike in previous attacks, this known knowledge of plaintext is not the leakage of

the unencrypted dataset, but rather the injected files created by the attackers themselves.

In the performance analysis of file injection attacks, the focus shifts from examining the recovery rate under different data leakage levels to evaluating the size of the injected document set corresponding to the size of the target keyword universe ($n$).

### 3.2.1. Binary Search attack

The attack described in [44] utilizes the concept of binary search to significantly reduce the required number of injected documents.

The way that the attacker of Binary search attack generates injected file is by letting file $i$ consisting of keywords from $K$ whose $i$th bit is 1. For instance, let's consider an example (Figure 3.1) with four keywords represented by $k_0, k_1, k_2$ and $k_3$. The highest index keyword is 3 and can be represented by a 2-digit binary number. In this example, file 1 is created by including keywords $k_2$ and $k_3$, as the first bit of the binary representation of 1 is 1. Similarly, file 2 consists of $k_1$ and $k_3$, as the second bit is 1.

By analyzing the combination of returned encrypted injected documents, the attacker can deduce the queried keyword with 100% accuracy of query recovery when no countermeasure is applied. For instance, if the return of a query contains only the encrypted File 1, the attacker can directly deduce that the underlying keyword of the query is $k_2$.

| | $k_0$ | $k_1$ | $k_2$ | $k_3$ |
|---|---|---|---|---|
| **File 1** | 0 | 0 | 1 | 1 |
| **File 2** | 0 | 1 | 0 | 1 |

**Figure 3.1:** Example of Binary Search Attack

Reducing the required number of injected documents from $n$ in [8] to $\lceil \log n \rceil$ is a significant achievement in the binary search attack.

However, it is essential to consider that each injected file in this approach contains exactly half of the keywords from the injected keyword universe. This characteristic can lead to a potential problem when the target keyword universe for recovery is of giant size, and the injected documents may become easily detectable due to their abnormal volume. This countermeasure is called threshold countermeasure.

### 3.2.2. Hierarchical Search attack

The Hierarchical Search attack, proposed by Zhang et al. [44], is a solution aimed at overcoming the threshold countermeasure used to resist the binary search attack.

This attack adopts a strategy of dividing the keyword universe into subsets and applies the binary search attack to each two of these subsets. By doing so, the attack achieves a balance between the size of injected files and the accuracy of query recovery. As a result, a set of injected files is generated, with a maximum size of $\lceil \frac{n}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$, where $T$ is the threshold of the maximal acceptable document length.

Regarding countermeasures, the paper also discusses the potential use of padding as a defense against the Hierarchical Search attack. However, at the time of their research in 2016, implementing padding in dynamic SSE schemes proved to be challenging. Therefore the researchers found that padding had no significant impact on mitigating the attacks they conducted.

### 3.2.3. Decoding attack

The decoding attack is noteworthy as it represents the first file injection attack that utilizes volume pattern information.

In this attack, the attacker first observes the corresponding total volume for all the queries, which serves as the baseline. The attacker then selects an offset value, denoted as $\gamma$, that cannot be evenly divided by the difference between the baseline volumes of any two queries.

For each keyword $k_i$, the attacker designs an injected document with a volume of $i \cdot \gamma$, filling it with repeating instances of the keyword $k_i$.

During the recovery phase, the attacker once again collects the corresponding total volumes of the queries and compares them with the baseline to deduce the underlying keyword for each query. For example, the difference between the new observed volume and the baseline volume of query $q_j$ is denoted as $v_j$. The attacker then searches for an integer $u$ such that $v_j = u \cdot \gamma$. If such an integer $u$ exists, the attacker concludes that the underlying keyword for $q_j$ is $k_u$.

In the context of DSSE with forward secrecy, the attacker may encounter a difficulty in matching $q_j$ with $v_j$, leading to longer computation times to determine whether a baseline volume exists that has a difference of $u \cdot \gamma$ from the observed total volume.

The expected number of injected files in file injection attacks is equal to the size of the target keyword universe that the attacker aims to recover.

Dynamic volume-hiding countermeasures can be applied to counter this attack. Additionally, the data collection of total volumes for all queries during the preparation step and the calculation of $\gamma$ already present significant difficulties for the practical implementation of this attack.

### 3.2.4. BVA attack

In 2023, Zhang et al. introduced a novel file injection attack BVA that leverages volume patterns, which is an extension of the ideas used in both the Decoding and Binary Search attacks.

In the BVA attack, before generating the injected documents, the attacker first observes the total volume of all queries. They then choose an injection parameter $\gamma$ such that $\gamma \geq \frac{n}{2}$, where $n$ represents the size of the target keyword universe.

Each injected file $f_i$ has a length of $\frac{n}{2}$ and contains keywords whose $i$th bit is equal to 1. Additionally, the volume size of $f_i$ is determined as $\gamma \cdot i^{i-1}$. When the keyword $w_i$ is queried, the total response size of the injected files is $\gamma \cdot i$.

The number of injected files in the BVA attack is the same as in the Binary Search attack, which is $\lceil \log n \rceil$. This represents a significant reduction compared to the previous file injection attack that used volume patterns - the Decoding attack. However, this reduction in the number of injected files comes at the cost of a slightly lower query recovery rate. In the worst case scenario, the BVA attack achieves a query recovery rate of 70%.

## 3.3. Conclusion

After a comprehensive analysis of the 13 SSE attacks, we have compiled a summary of the key properties of each attack in Table 3.1. The attacks are categorized into two groups: inference attacks and file injection attacks. In this section, we would like to conclude our understanding.

### 3.3.1. Inference Attack

Through our investigation of inference attacks, it has become evident that the final query recovery rate is significantly influenced by the attacker's precision in identifying keywords. The evolution of inference attacks, from the early IKK to the recent VAL attack, showcases the continuous advancements in matching methods between keywords and queries.

In the initial stages of inference attacks, like IKK, emphasis was placed on utilizing co-occurrence probability matrices to establish associations between keywords and queries. However, as research progressed, the most cutting-edge technique emerged, focusing on the occurrence positions among matched documents to define keywords. This shift in approach led to improved precision and efficiency in inference attacks with partial data leakage, as it involved first matching documents and subsequently using those matches as references for query recovery, rather than directly matching queries.

**Table 3.1:** Overview of SSE Attacks

| **IN Attack**[1] | **Leakage Pattern** | **Data Leakage** | **Approach** | **Target** |
|---|---|---|---|---|
| IKK [17] | AP | All | Keyword Co-occurrence | Query |
| Count [8] | AP,RLP | Partial (sizeable) | Keyword Co-occurrence, Response length | Query |
| Shadow Nemesis [31] | AP | Similar | Keyword Co-occurrence | Query |
| VA [3] | TVP | Partial | Document volume | Query |
| SVA [3] | VP | Partial | Document volume, Response length | Query |
| Subgraph [3] | ALP | Partial | Subset | Query |
| NXLZC [29] | AP | Partial | Length | Document, Query |
| LEAP [28] | AP | Partial | Document Co-occurrence, Length | Document, Query |
| VAL [24] | AP,VP | Partial | Document Co-occurrence, Length, Volume | Document, Query |
| **FI Attack**[1] | **Leakage Pattern** | **Injection Size**[2] | | **Target** |
| Basic FI [8] | AP | $n$ | | Query |
| Binary Search [44] | AP | $\lceil \log n \rceil$ | | Query |
| Hierarchical Search [44] | AP | $\lceil \frac{n}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$ | | Query |
| Decoding [3] | VP | $n$ | | Query |
| BVA [43] | VP | $\lceil \log n \rceil$ | | Query |

[1] 'IN' represents inference attacks, while 'FI' stands for file injection attacks. Inference attacks are evaluated based on the size of data leakage and the approach used for inference, and file injection attacks are compared based on the amount or number of injected files.
[2] $n$ denotes the number of injected keywords and $T$ means the length limitation of each document.

As we reviewed each attack, we recognized that they each represent significant milestones, introducing novel ideas and approaches to the field of inference attacks. These advancements build upon one another, collectively enriching our understanding of the vulnerabilities present in secure search schemes and highlighting the need for robust countermeasures.

Moreover, an intriguing observation is that the frequency of the chosen keyword universe does influence the final results, although the exact reasons behind this phenomenon remain unclear.

### 3.3.2. File Injection Attack

The field of file injection attacks has made significant strides in minimizing the number of injected documents, and adapting the attacks to different leakage environments, including scenarios where access patterns are not available, such as when ORAM is used in searchable encryption.

File injection attacks have a significant advantage in achieving 100% query recovery with no data leakage since their inception. However, this advantage also implies that the scope for further improvement in subsequent file injection attacks is relatively limited compared to inference attacks.

The publication of the Binary Search attack marked a breakthrough in reducing the number of injected documents required for the attack. Subsequent attacks, such as Decoding and BVA, have explored the use of volume patterns, further enhancing the practicality and efficiency of file injection attacks.

However, during the study of BVA, a trade-off was observed between the number of injected documents and the final query recovery rate.

Overall, the advancements in file injection attacks have been impressive, with researchers continuously striving to find the best balance between the number of injected documents and the query recovery rate while adapting the attacks to diverse leakage scenarios.

## 3.4. Intuition of Improvement

After summarizing the current development of SSE attacks, several concrete insights have emerged that can help address the abstract research question of 'designing a new SSE attack that exhibits better performance or greater practicality.'

### 3.4.1. Inference Attack

There are two insights regarding potential improvements in inference attacks:

**Wider Keyword Universe Selection:** Research, such as [3], has shown that the choice of keywords in the keyword universe, specifically their occurrence frequency, significantly impacts the final query recovery rate. However, this aspect has not been extensively explored beyond the mentioned research. Considering that low-frequency keywords may also contain valuable and sensitive information in real-life scenarios, it would be intriguing to investigate the extent to which the query recovery rate can be improved when using a wider range of keywords, including low-frequency ones.

**Partial of Partial Data Leakage:** The second area of improvement involves reducing the assumptions made in inference attacks. The VAL attack showcased that a high query recovery rate can be achieved with as little as 10% of data leakage, where data leakage refers to the number of leaked documents. However, an unexplored territory lies in scenarios where only a partial portion of a document is leaked. The question arises: Can attackers still reveal sensitive content with such limited information? This concept of 'partial of partial data leakage' presents a compelling and untapped area for investigation.

### 3.4.2. File Injection Attack

**Allowing for Partial Data Leakage:** The current file injection attacks have demonstrated impressive query recovery rates without the need for data leakage. However, as discussed in the previous chapters, completely avoiding data leakage in an encryption system is challenging due to potential human factors and other practical considerations. An intriguing research direction for file injection attacks is to explore scenarios where partial data leakage is permitted. In such cases, the attack could utilize a small amount of partial data leakage to achieve its objectives. The question then arises: Can file injection attacks effectively reduce the number of injected documents when there is only a small amount of partial data leakage available?

**Reducing the impacts of Countermeasure:** Another crucial research direction for both inference and file injection attacks is finding ways to overcome the resistance posed by access or volume hiding countermeasures. As SSE systems continue to improve their defenses, attackers will also develop more robust and sophisticated attack strategies to remain effective. The question of whether attacks can be equipped with stronger abilities to counteract these countermeasures is an ongoing and ever-evolving research area.

### 3.4.3. Our Direction

Our research will be dedicated to exploring the direction of allowing partial data leakage in file injection attacks, with the aim of developing a novel attack that can further reduce the number of injected documents required.

While the current approach to reducing the number of injected documents involves finding methods

to insert more keywords into each injected document, this approach has its limitations. The Binary Search attack, which efficiently recovers $n$ keywords using only $\lceil \log n \rceil$ injected documents, has set a challenging record that is difficult to surpass using traditional logic.

Inspired by the 'document recovery first, then query recovery' concept utilized in the most recent inference attacks such as LEAP and VAL attacks, we intend to incorporate this idea into existing file injection attacks. The goal is to provide additional information to each injected keyword by linking them to leaked documents. This innovative approach has the potential to recover the same amount of keywords with fewer injected documents, in other words, creating an amplifier for query recovery.

In the upcoming chapter, we will present a detailed procedure of our proposed attack, along with corresponding experimental results.

# 4

# Initial Idea

In this chapter, we present our initial attempt to build an amplifier for file injection attacks, with the goal of reducing the size of injected documents required for query recovery. We will provide a comprehensive description of our proposed design and the results of the experiment.

## 4.1. Notation

We present Table 4.1, which comprises the notations used in our attack, along with their explanations:

**Table 4.1:** Summary of notations used in our attack.

| Notations | Definition |
|---|---|
| $D$ | Plaintext document set, $D = \{d_1, ..., d_n\}$ |
| $ED$ | Server document set, $ED = \{ed_1, ..., ed_n\}$ |
| $D'$ | Leaked document set, $D = \{d_1, ..., d_{n'}\}$ |
| $W$ | Keyword universe, $W = \{k_1, ..., k_m\}$ |
| $Q$ | Query set, $Q = \{q_1, ..., q_m\}$ |
| $W'$ | Known Keyword set, $W' = \{k_1, ..., k_{m'}\}, W' \in W$ |
| $W'_i$ | Chosen keyword set, $W' = \{k_1, ..., k_{m'_i}\}, W'_i \in W'$ |
| $IF$ | Inject file set, $IF = \{id_1, ..., id_k\}$ |
| $EIF$ | Encrypted injected file set, $EIF = \{eid_1, ..., eid_k\}$ |
| $T$ | Limitation of the number of keywords in each document |
| $\|d_i\|$ | Number of keywords/queries in $d_i$ |
| $\|D\|$ | Number of files in $D$ |
| $\alpha$ | Number of keywords selected from each document |
| $C$ | Set of matched documents |
| $C_i$ | Set of matched injected documents |
| $R$ | Set of matched queries |
| $R_i$ | Set of matched queries by file injection |

## 4.2. The Design

In this attack, the attacker possesses the same capabilities as in previous file injection attacks, where they can deceive the user into uploading the pre-designed injected files. The primary objective of the attack is to recover as many queries as possible, where the underlying keywords are in the set of known keywords extracted from leaked documents.

### 4.2.1. Leakage Model

In our proposed attack, the attacker is assumed to have access to the following leakage:

A set of leaked documents $D' = \{d_1, d_2, ..., d_{n'}\}$, where $D'$ is a subset of the plaintext document set $D = \{d_1, d_2, ..., d_n\}$. Here, $n'$ represents the number of leaked documents, and $n$ is the total number of documents in the dataset, $n' \leq n$. It is assumed that the attacker has access to a portion of the original documents.

A set of leaked keywords $W' = k_1, k_2, ..., k_{m'}$, which can be extracted from the leaked documents $D'$. $W'$ is a subset of the entire keyword universe $W = k_1, k_2, ..., k_m$ of the original dataset $D$. Here, $m'$ denotes the number of known keywords, and $m$ is the size of the entire keyword universe, and $m' \leq m$.

The attacker can intercept the communication between the user and the server, enabling them to obtain the access pattern that explains the occurrence relationship between the encrypted dataset $ED = ed_1, ed_2, ..., ed_n$ and the queries $Q = q_1, q_2, ..., q_m$ (Level 2). Here, $ED$ and $Q$ represent the encrypted versions of $D$ and $W$, respectively.

Additionally, the attacker is assumed to have knowledge of the documents matching between the injected documents $IF = \{id_1, id_2, ..., id_k\}$ and their corresponding encrypted versions $EIF = \{eid_1, eid_2, ..., eid_k\}$.

To achieve this assumption, the attacker may employ methods such as matching the identifiers of documents by observing the following uploaded encrypted documents with the same length as the injected documents.

### 4.2.2. Intuition

The core idea behind this attack is to 'embed more meaning into each keyword in the injected documents.' This concept is based on the hypothesis that low-frequency keywords can be utilized to identify leaked documents.

To better understand where this hypothesis comes from, let's consider an example in an ideal scenario:

Imagine that Alice sends four emails to different recipients, arranging meetings with them:

> Email 1 (Sent to Bob):
> Content: Hi **Bob**, I have scheduled a meeting for the midterm evaluation. The specific location and time are...
>
> Email 2 (Sent to John):
> Content: Hi **John**, we need to plan a half-hour meeting to discuss important updates and changes in the project.
>
> Email 3 (Sent to Anna):
> Content: Hi **Anna**, thank you for inviting me to your graduation party. I am delighted to attend and would like to confirm the exact time with you...
>
> Email 4 (Sent to Smith):
> Content: Dear Professor **Smith**, I have a question about the assignment ...

Those emails will be then encrypted and uploaded to a server and deleted from the local device. When Alice later wants to search for a certain email, she is most likely to use the keywords that differentiate the four emails, such as the words marked in bold in each email which is the name of the recipients. This is a common approach used in real-life scenarios when conducting precise document searches: we select keywords that provide the strongest unique identity to the target document.

The attacker can apply a similar logic during the inject keyword selection process. Instead of using the entire known keyword set $W'$ (the target to be recovered), the attack can choose the least frequent keywords from the leaked documents and label them as the injected keyword set $W'_i =$

$\{Bob, John, Anna, Smith\}$. Here, $W_i'$ is a subset of $W'$ and has a size equal to the number of leaked documents, as one least frequent keyword is selected from each document.

Next, the attacker generates injected documents using $W_i'$, using methods proposed by existing file injection attacks like the Binary Search attack, which ensures a 100% query recovery accuracy with the current smallest possible injected file size. In this case, with four keywords in $W_i'$, only two injected documents are needed: $id_1 = \{Anna, Smith\}$ and $id_2 = \{John, Smith\}$. The attacker is assumed to know $eid_1$ and $eid_2$ are the encryption versions of $id_1$ and $id_2$, respectively.

After recovering $W_i'$, the attacker can identify the corresponding leaked documents. For instance, if query $q_1$ has a server response of $\{eid_1, ed_3\}$, the attacker can infer that the underlying keyword of $q_1$ is $Anna$, and the corresponding leaked document of $ed_3$ is "Email 3."

By comparing the unique occurrence patterns among those matched documents, the attacker can recover additional keywords in $W'$ that are not present in $W_i'$.

In summary, this approach leverages the power of low-frequency keywords to amplify the query recovery capability of the attack.

It is crucial to acknowledge that the scenario proposed above is based on an ideal assumption, that a document can be uniquely identified by a single keyword. However, as we widen our perspective beyond the leaked documents, we realize that they represent only a small portion of the entire dataset $D$. Relying solely on a single keyword to identify a document becomes challenging since there could be other documents within the dataset that contain the same keyword but have not been leaked. Such as the example given below, where green color indicates the non-leaked documents:

Email 1 (Sent to Bob):
Content: Hi Bob, I have scheduled a meeting for the midterm evaluation. The specific location and time are...

Email 2 (Sent to John):
Content: Hi John, we need to plan a half-hour meeting to discuss important updates and changes in the project.

Email 3 (Sent to Anna):
Content: Hi Anna, thank you for inviting me to your graduation party. I am delighted to attend and would like to confirm the exact time with you...

Email 4 (Sent to Smith):
Content: Dear Professor Smith, I have a question about the assignment ...

Email 5 (Sent to Bob):
Content: could you please let me know when are you free for the next meeting.

Email 6 (Sent to Anna):
Content: Hi Anna, the last form you still have to fill out before graduation is...

...

Using only the names of recipients as keywords, such as 'Bob' and 'Anna', may no longer be sufficient to accurately identify the leaked documents, as these names could be duplicated in multiple documents within the dataset $D$. Consequently, the accuracy of document recovery is reduced, which ultimately impacts the final query recovery rate.

To address this challenge, we propose increasing the number of keywords chosen from each document. To represent the number of keywords chosen from each document, we will use the symbol $\alpha$. A detailed description of the procedure is provided in the following section.

### 4.2.3. Procedure

The attack model's overview, as shown in Figure 4.1, consists of three main procedures: keyword selection, file injection, and recovery.

By delving into a more detailed hierarchical breakdown of these procedures, we can further divide the recovery procedure into sub-procedures. For instance, the initial step involves the recovery of injected keywords. Subsequently, the document recovery process begins, utilizing the recovered injected keywords to identify leaked documents and their corresponding queries. Finally, the matched documents are used to recover additional keywords in $W'$ that are not present in $W'_i$.

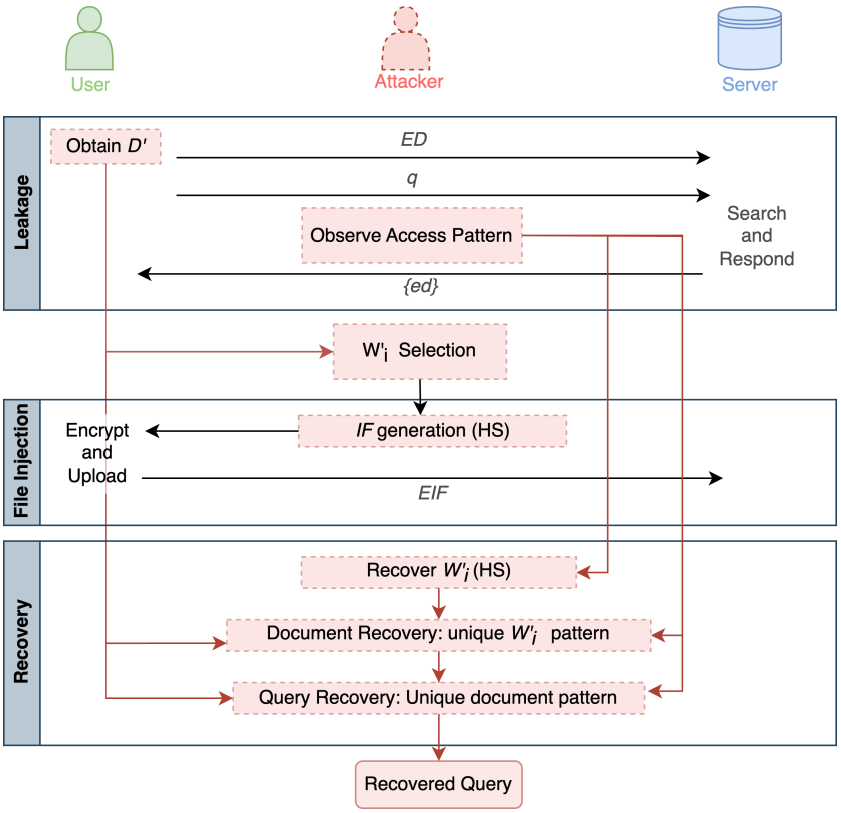



**Figure 4.1:** Attack model

#### Keyword Selection

During this step, the attacker takes the leaked documents $D'$ as input and extracts meaningful keywords from each of these leaked documents. The attacker then includes $\alpha$ of these extracted keywords into the set of injected keywords. The value of $\alpha$ is determined experimentally.

The strategy used to choose these $\alpha$ keywords may significantly influence the recovery results for both document and query recovery. As a result of this consideration, we have proposed three different strategies to address this aspect of the attack:

**V0. Random Choose:** In this strategy, the attacker randomly selects $\alpha$ keywords from each leaked document, which may include both high and low frequent keywords.

**Algorithm 1** Keyword_Selection(V0)

**Input**: A set of known document $D'$ and a variable $\alpha$
**Output**: A set of chosen keyword for injection $W_i'$

**procedure** Keyword_Selection($D$', $\alpha$)
    Initialize an empty list $W_i'$
    **for** $i = 1 \rightarrow$ #$D$' **do**
        $k \leftarrow \alpha$ randomly chosen keywords in $d_i$
        $W_i' \leftarrow W_i' \cup k$ ▷ excluding the repetition
    **end for**
    **return** $W_i'$
**end procedure**

**V1. Local Least Frequent:** In this strategy, the attacker sorts keywords extracted from each document in frequent order and chooses $\alpha$ of them with the least frequent keywords. The selected keywords are guaranteed to have the least frequency within their respective documents, but they might have a relatively high frequency among the entire known keyword set.

**Algorithm 2** Keyword_Selection(V1)

**Input**: A set of known document $D'$ and a variable $\alpha$
**Output**: A set of chosen keyword for injection $W_i'$

**procedure** Keyword_Selection($D'$, $\alpha$)
    Initialize an empty list $W_i'$
    **for** $i = 1 \rightarrow$ #$D'$ **do**
        $k \leftarrow \alpha$ least frequent keywords in $d_i$
        $W_i' \leftarrow W_i' \cup k$ ▷ excluding the repetition
    **end for**
    **return** $W_i'$
**end procedure**

**V2. Global Least Frequent:** In this strategy, the attacker gathers all known keywords first and sorts them in order of frequency based on the sum count of occurrences among the leaked documents. Then, for each document, the attacker sorts its keywords based on the obtained global rank and selects $\alpha$ with the lowest global frequency. This method is expected to have a better ability to identify documents; however, the size of the injected keyword set might be larger compared to the other two strategies.

**Algorithm 3** Keyword_Selection(V2)

**Input**: A set of known document $D'$ and corresponding set of known keyword $W'$, and a variable $\alpha$
**Output**: A set of chosen keyword for injection $W_i'$

**procedure** Keyword_Selection($D$', $W'$, $\alpha$)
    Sort $W'$ based on the occurrence count of keywords among $D'$
    Initialize an empty list $W_i'$
    **for** $i = 1 \rightarrow$ #$D$ **do**
        $k \leftarrow \alpha$ keywords in $d_i$ which have the least global frequent in $W'$
        $W_i' \leftarrow W_i' \cup k$ ▷ excluding the repetition
    **end for**
    **return** $W_i'$
**end procedure**

The size of $W_i'$ is supposed to be no larger than $\alpha \cdot n'$.

#### File Injection and Recovery

During this step, we have opted to employ an existing file injection attack that utilizes the access pattern as a black box to aid in the process. The two options we considered were the Binary Search attack and the Hierarchical Search Attack (HSA). After a comprehensive evaluation, we have chosen the HSA for this purpose.

The main reason behind this choice is twofold:

From our perspective, the HSA offers practical advantages, particularly in its ability to resist the threshold countermeasure. This resilience is necessary for the overall effectiveness and robustness of our attack, as it ensures that our attack also inherits this resistance to the threshold countermeasure and maintains stable performance.

From the perspective of the HSA, while the HSA exhibits superior resistance to countermeasures, it has the drawback of sacrificing the ability to reduce the number of injected documents compared to the Binary Search attack. However, our innovative approach is expected to enhance the HSA's performance in terms of further reducing the number of injected documents. This enhancement makes the HSA more practical and efficient in real-world scenarios, and the trade-off between reduced injected documents and improved resistance to countermeasures aligns well with our attack objectives.

Given the selected keywords $W_i'$ and a threshold length of each injected file $T$, the injection algorithm of HSA will produce a set of files for injection, denoted as $IF$. The maximum size of $IF$ is determined by the formula:

$$|IF| = \lceil \frac{\alpha \cdot n'}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$$

After injecting $IF$ into the server database, the attacker closely monitors the arrival time and size of the newly injected files. By analyzing this information, the attacker can establish a set of documents matching $C_i$, which contains the correspondence between the injected files $IF$ and the server files $EIF$.

Next, the attacker inputs $C_i$, $W_i'$, and the collected access patterns between $ED$ and $Q$ into the recovery algorithm of the HSA. The recovered query set is denoted as $R_i$. Since HSA is used as a black box in this recovery process, the output is expected to achieve 100% accuracy, meaning that $|R_i| = |W_i'|$.

#### Document Recovery

With the current set of matched keywords $R_i$, it becomes possible to identify the underlying leaked documents of encrypted documents by examining their access patterns.

Each document can be characterized by a set of recovered queries, and we define that a document $d_i$ is a candidate for $ed_i$ if they share the same set of queries.

In this attack, we focus on a single match scenario, which means that if there is only one candidate in the set, we consider $ed_i$ to be the underlying encrypted document of $d_i$ and obtain the mapped document set $C$.

#### Remaining Keyword Recovery

During this step, the primary objective is to recover as many queries as possible that contain underlying keywords from $W'$ but are not present in $W_i'$. As mentioned before, we would like to reach this objective by using the occurrence pattern of keywords as the reference.

Algorithmically, this step can be viewed as a reversed version of the algorithm used for document recovery. For each keyword $k_i \in W'$ and $k_i \notin W_i'$, we define it by a set of documents that have been matched where the keyword is contained in those documents. Then, it is uniquely matched with a query that can be defined by the same set of documents.

By merging the newly matched keywords with $R_i$, we obtain the final set of mapped queries in the attack, denoted as $R$.

### 4.2.4. Pseudocode of the Proposed Attack

When all the individual phases are combined, we obtain an overview of the entire procedure of the proposed attack. We present this overview in Algorithm 6.

**Algorithm 4** Document_Recovery

**Input**: A set of known document $D'$, a set of server document $ED$, a set of mapped queries by file injection $R_i$
**Output**: A set of mapped document $C$

Initialize an empty dictionary $C$
**for** $ed_i \in ED$ **do**
    $ed_i \leftarrow \{q_1, ...q_a\}$ if $q \in ed_i$, for $q \in R_i$
**end for**
**for** $d_i \in D'$ **do**
    $d_i \leftarrow \{q_1, ...q_b\}$ if $q \in d_i$, for $q \in R_i$
**end for**
**for** $d_i \in D'$ **do**
    candidates $\leftarrow ed_j$ for $j \in [n]$ where $\{q_1, ...q_b\} == \{q_1, ...q_a\}$
    **if** $|candidates| == 1$ **then**
        $C[candidates[0]] \leftarrow d_i$
    **end if**
**end for**
**return** $C$

**Algorithm 5** Remaining_Keyword_Recovery

**Input**: A set of mapped document $C$, a set of query $Q$, a set of known keywords$W'$, a set of mapped queries $R_i$
**Output**: A set of mapped queries $R$

**for** $w_i \in W'$ **do**
    **if** $w_i \in R_i$ **then**
        remove $w_i$ from $W'$
    **else**
        $w_i \leftarrow \{d_1, ...d_\alpha\}$ if $W_i \in d$, for $d \in C$
    **end if**
**end for**
**for** $q_i \in Q$ **do**
    **if** $q_i \in R_i$ **then**
        remove $q_i$ from $Q$
    **else**
        $q_i \leftarrow \{d_1, ...d_\alpha\}$ if $q_i \in d$, for $d \in C$
    **end if**
**end for**
**for** $w_i \in W'$ **do**
    candidates $\leftarrow q_j$ for $j \in [|Q|]$ where $\{d_1, ...d_\alpha\} == \{d_1, ...d_\alpha\}$
    **if** $|candidates| == 1$ **then**
        $R[candidates[0]] \leftarrow w_i$
    **end if**
**end for**
$R \leftarrow R \cup R_i$
**return** $R$

## 4.3. Countermeasure Discussion

The proposed attack combines the characteristics of both inference attacks and file injection attacks: the Hierarchical Search Attack is employed to recover the initial matched queries, which are later used for document and query recovery in an inference method. This makes the attack a unique and potentially dangerous threat. To mitigate its impact, we will discuss the extent to which performance can be mitigated by employing both static and dynamic countermeasures against inference and file injection attacks, respectively.

As mentioned in the previous chapter, the countermeasures used to conceal access patterns are

**Algorithm 6** New Attack

**Input**: A set of known document $D'$ and corresponding set of known keyword $W'$
**Output**: A set of mapped query $R$

**procedure** Keyword_Selection($D'$, $\alpha$)
    Initialize an empty list $W_i'$
    $W_i' \leftarrow$ Keyword_Selection$(D', \alpha)$ ▷ Alg. 1/2/3
    **return** $W_i'$
**end procedure**

**procedure** Injection($W_i'$, $T$)
    $IF \leftarrow$ Inject_Files_hierarchical$(W_i', T)$ ▷ file injection algorithm of HSA
    **return** $IF$
**end procedure**

Inject $IF$ and observe $Q$, $ED$ and $C_i$

**procedure** Recovery($Q$, $ED$, $W'$, $C_i$, $W_i'$)
    $R_i \leftarrow$ Recover_hierarchical$(C_i, W_i')$ ▷ Recovery algorithm of HSA
    $C \leftarrow$ Document_Recovery$(ED, D', R_i)$ ▷ Alg. 4
    $R \leftarrow$ Remaining_Keyword_Recovery$(C, Q, W')$ ▷ Alg. 5
    **return** $R$
**end procedure**

obfuscation and padding. These measures return misleading corresponding encrypted documents in the server's response to queries, distorting the relationship between queries and server documents, and confusing the attacker. Our discussion will be divided into the influence caused under the application of static or dynamic padding and obfuscation.

If we consider the static padding and obfuscation measures [10, 34], we conclude that they have a relatively weaker influence on our attack since they only apply during the setup of the schemes. For the attacker, the observed access pattern of injected files remains unaffected.

However, these measures still impact the document matching phase, which relies on the unique presence of a specific combination of queries. Padding may increase the number of candidates for documents sharing the same query combination, while obfuscation can lead to incorrect or no matches, reducing document recovery accuracy, and potentially affecting query recovery results.

Nevertheless, by using static padding and obfuscation, the final recovered queries are equal to the number of injected keywords in the worst case scenario, which offers no improvement compared to using only the Hierarchical Search Attack.

However, dynamic padding and obfuscation [32, 41] introduces the practice of updating the bogus access pattern for each update, which has a notable impact on our attack. By constantly modifying the bogus access pattern, dynamic padding disrupts the accurate recovery of queries even during the file injection phase. As a result, both query recovery and document recovery are negatively affected.

Apart from dynamic padding and obfuscation, other countermeasures specifically designed to counter file injection attacks can also significantly reduce the performance of our attack. One such example is the Vaccine technique [25], which has been previously mentioned.

## 4.4. Experiment

In this section, we will provide an overview of the experimental setup, including the dataset used and the data processing steps. Following that, we will introduce the benchmark used to evaluate the performance of the proposed attack. Finally, we will present and analyze the results obtained from the experiments.

### 4.4.1. Setup

#### Dataset

The performance of the proposed attack was evaluated through tests conducted on three real-life datasets that were also used by VAL attack: Enron [26], Lucene [14], and Wikipedia. These datasets were chosen to gain insight into how the attack operates under different data types. Table 4.2 presents a summary of the datasets used, providing a brief overview of each.

**Table 4.2:** Scales of Datasets

| Dataset | Enron | Lucene | Wikipedia |
|---|---|---|---|
| **No. of documents** | 30,109 | 51,317 | 20,000 |
| **No. of keywords** | 63,029 | 92,976 | 148,367 |

**Enron:** The Enron dataset is a publicly available collection of email communications generated by 150 senior managers of Enron Corporation. For our evaluation, we specifically selected the emails from the $\_sent\_mail$ folder, which consists of 30,109 emails. We use the content of these emails as the plaintext representation of the documents in $D$. Figure 4.2 provides an example of a document of this dataset.

**Lucene:** The Lucene dataset is an email listing dataset that captures communication between users and PyLucene Developers. For our evaluation, we focused on the ”java-user” category, which is dedicated to addressing user issues. The selected dataset covers a period from 2001 to 2011, with data available from September to December in 2001. It includes a total of 51,317 emails. Similar to the Enron dataset, we extract the content of the emails to be the plaintext representation of our documents in $D$.

The Python code used to download the dataset is provided in Appendix A, and the emails of each month are compressed as a mbox file. An example of a Lucene email is shown in Figure 4.3.

**Wikipedia:** Wikipedia is a widely recognized online encyclopedia created and maintained by a community of volunteers. For our evaluation, we obtained a subset of 20,000 articles from a simple wiki dump provided by David Shapiro [35].

The dataset has a size of approximately 1.19 GB after decompression[1]. An example of a Wikipedia article from this dataset is shown in Figure 4.4.

#### Preparing the Dataset

Given that this is the experiment of the initial attempt in our project, we opted to utilize 50% of each dataset to rapidly obtain a comparative performance analysis of the three strategies. The selection of documents in each dataset was done randomly.

We preprocessed these datasets to retain only meaningful keywords by removing the most common English stopwords, such as ”a,” ”the,” ”is,” and others, using the NLTK package [38]. The number of remaining keywords for each dataset is listed in Table 4.2. This preprocessing step helps filter out frequently occurring words that do not contribute significantly to the analysis.

After removing stopwords, we further improve the quality of the remaining keywords by applying stemming using the Porter Stemmer algorithm [30]. Stemming involves reducing words to their root or base form to unify similar words that may have different variations due to tense or pluralization.

Document $D$ is all documents in each dataset. Unlike previous SSE attacks, we decided to run the attack in a broader keyword universe, where all keywords that can be extracted from the documents are considered. The target for recovery is the keyword set $W'$ extracted from the leaked documents $D'$. By doing so, we expect the results to be less influenced by the keyword frequency, as mentioned in [3].

To evaluate the impact of the size of the leaked dataset on document and query recovery results, we emulate the settings of the VAL attack and execute the proposed attack with different levels of data leakage, specifically at 0.1%, 0.5%, 1%, 5%, and 10% of the total dataset. Please note that we do not

[1] https://dumps.wikimedia.org/simplewiki/latest/simplewiki-latest-pages-articles-multistream.xml.bz2

Message-ID: <14015272.1075856157694.JavaMail.evans@thyme>
Date: Fri, 8 Sep 2000 08:35:00 -0700 (PDT)
From: tom.donohoe@enron.com
To: susan.elledge@enron.com
Subject: more contracts
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-From: Tom Donohoe
X-To: Susan Elledge
X-cc:
X-bcc:
X-Folder: /Thomas_Donohoe_Dec2000/Notes Folders/'sent mail
X-Origin: Donohoe-T
X-FileName: tdonoho.nsf

I am in need of several more contract copies. Could you please forward me
contract:
96037261
96049419
96001003
96045391

Also, you had mentioned earlier that GISB contracts should not have been used on some of the contracts I earlier received. Are the appropriate contracts going to be sent out, or are we stuck with the ones in place.

Call with questions. 3-7151

Sincerely,
Tom Donohoe

**Figure 4.2:** Sample Document of Enron

From MAILER-DAEMON Fri Jul 31 10:51:21 2009
Return-Path:
<java-user-return-41576-apmail-lucene-java-user-archive=lucene.apache.org@lucene.apache.org>
Delivered-To: apmail-lucene-java-user-archive@www.apache.org
Received: (qmail 19601 invoked from network); 31 Jul 2009 10:51:21 -0000
Received: from hermes.apache.org (HELO mail.apache.org) (140.211.11.3)
by minotaur.apache.org with SMTP; 31 Jul 2009 10:51:21 -0000
Received: (qmail 19232 invoked by uid 500); 31 Jul 2009 10:03:36 -0000
Delivered-To: apmail-lucene-java-user-archive@lucene.apache.org
Received: (qmail 19170 invoked by uid 500); 31 Jul 2009 10:02:52 -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 19095 invoked by uid 99); 31 Jul 2009 10:02:13 -0000
Received: from nike.apache.org (HELO nike.apache.org) (192.87.106.230)
by apache.org (qpsmtpd/0.29) with ESMTP; Fri, 31 Jul 2009 10:02:08 +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 lists@nabble.com designates
216.139.236.158 as permitted sender)
Received: from [216.139.236.158] (HELO kuber.nabble.com) (216.139.236.158)
by apache.org (qpsmtpd/0.29) with ESMTP; Fri, 31 Jul 2009 10:01:57 +0000
Received: from isper.nabble.com ([192.168.236.156])
by kuber.nabble.com with esmtp (Exim 4.63)
(envelope-from <lists@nabble.com>)
id 1MWovg-0001bo-Vs
for java-user@lucene.apache.org; Fri, 31 Jul 2009 03:01:36 -0700
Message-ID: <24753954.post@talk.nabble.com>
Date: Fri, 31 Jul 2009 03:01:36 -0700 (PDT)
From: bourne71 <garylkc@live.com>
To: java-user@lucene.apache.org
Subject: Boosting Search Results
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Nabble-From: garylkc@live.com
X-Virus-Checked: Checked by ClamAV on apache.org

Hi, new here.
I recently started using lucene and had encounter a problem.I crawl and index a number of documents.
When i perform a search, lets say ”tall fat”, by right the results that matches all the keyword should be on top and display first.
But in my search results, some of the document with only 1 matches of the keyword like ’tall’ is display first. Why is that? What had i done wrong?
can anyone advise me on this? thanks
–
View this message in context: http://www.nabble.com/Boosting-Search-Results-tp24753954p24753954.html
Sent from the Lucene - Java Users mailing list archive at Nabble.com.

---

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

**Figure 4.3:** Sample Document of Lucene

{
”id”: ”998052”,
”text”: ”V is a British five piece-boyband that was formed by several auditions in Prestige in 2003 which disbanded in 2005.
== Career ==
The band released four hit singles, their debut single ”Blood, Sweat & Tears” reached 6 in UK, second double A-side single”Can You Feel It”/”Hip To Hip” which reached 5 in UK & ”You Stood Up” which reached 12 in UK. Additional single ”Earth, Wind & Fire”, released in 2005, is also one of the hits by the band. Every singles make success with their studio album ”You Stood Up”.
== After disband ==
* Kevin is in gay relationship with Westlife member Markus Feehily and also a photographer
* Mark Harle became a drummer for indie band Little Comets until 2011, when he left the band.
* Aaron Buckingham moved into A&R; work, including managing the band Lawson.
* Antony Brant remained friends with the members of McFly, and toured with the band alongside James Bourne and the Vamps in the spring of 2013 as a compere to their Memory Lane tour.”,
”title”: ”V (group)”
}

**Figure 4.4:** Sample Document of Wikipedia

consider the 30% data leakage setting due to the extensive keyword universe used in our attack, which could result in considerably long running times for multiple repetitions of the experiment.

Additionally, we investigate the number of keywords required to correctly identify a document by running the attack with different $\alpha$ values: 2, 3, 5, and 10.

To ensure a robust assessment, each experiment will be conducted 10 times.

### 4.4.2. Evaluation Criterion

We will assess the performance of three strategies of the proposed attack based on their results in both document and query recovery.

The accuracy of document recovery indicates the proportion of leaked documents $D'$ that have been correctly recovered. It is measured as the ratio of correctly matched documents in the set $C$ to the total number of leaked documents, the injected document set is excluded:

$$ACC_d = \frac{|correct\ match\ in\ C|}{|D'|} \times 100\%$$

The accuracy of query recovery indicates how many queries have been correctly matched to their underlying keywords. This measure includes both queries recovered during the injection stage and those recovered during the remaining keywords recovery stage:

$$ACC_q = \frac{|correct\ match\ in\ R|}{|W'|} \times 100\%$$

In addition to the accuracy of document and query recovery, we introduce an additional criterion to analyze the amplifier effect achieved by the attack: the rate of return. This indicator evaluates, on average, how many keywords can be recovered by a single injected keyword. It is calculated as:

$$rate\ of\ return = \frac{|correct\ match\ in\ R|}{|W_i'|}$$

A higher rate of return indicates a more effective strategy in terms of query recovery capabilities.

### 4.4.3. Result

In the analysis of the results, we will focus on two aspects: document recovery and query recovery.

In the document recovery part, our aim is to determine if our attack can achieve comparable outcomes to existing inference attacks that involve document recovery by choosing $\alpha$ keywords to identify a document.

For the query recovery part, we will assess how many keywords can be correctly recovered within this broader keyword universe. Additionally, we will measure the amplifier effect we can create, which indicates how much the recovered query set expands beyond the initially injected keywords. This will provide insights into the effectiveness of our attack.

Document Recovery:

Figure 4.5 presents the performance of the three strategies (Random Choose - V0, Local Least Frequent - V1, and Global Least Frequent - V2) under various data leakage levels, with an $\alpha$ value of 5, which is the average of our range of $\alpha$, which is $\frac{2+3+5+10}{4} = 5$.

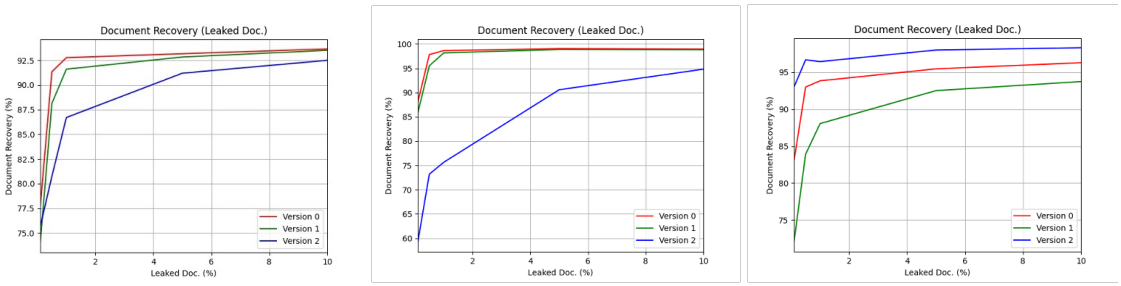
On the other hand, Figure 4.6 shows the outcomes achieved with different $\alpha$ values, and the data leakage level is $\frac{0.1+0.5+1+5+10}{5} \times 100\% = 3.32\%$.

When comparing the performance of the attack among the three datasets under various levels of data leakage (Figure 4.5), we observe that the performance of the three strategies is relatively similar in the Enron and Lucene datasets, but different in the Wikipedia dataset.

In the Enron and Lucene datasets, V0 consistently exhibits the best performance in document recovery, maintaining a stable high document recovery rate since a data leakage level of 1%. V1 follows closely behind V0 and also shows stable performance, with its document recovery rate nearly overlapping with V0's at data leakage levels of 1% and beyond. In Lucene, the document recovery rate of V1 is almost constantly on par with V0 starting from a data leakage of 1%.

However, V2 constantly lags behind V0 and V1, and its document recovery rate only stabilizes at a relatively high level starting from a data leakage of 5%.

However, in the Wikipedia dataset, the story is reversed. V2 becomes the strategy with constantly the best performance, followed by V0, and then V1. All three strategies show stable performance starting from a data leakage of 1% in this dataset.




**(a)** Enron **(b)** Lucene **(c)** Wiki

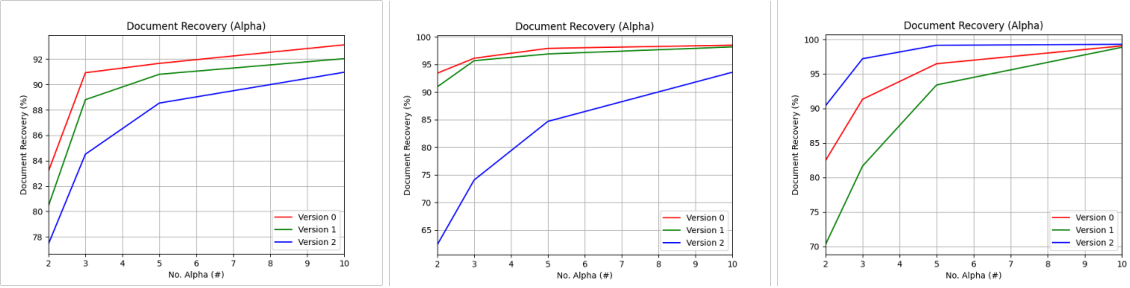
**Figure 4.5:** Document Recovery Performance (Leaked Doc%)

In Figure 4.6, a similar trend in the performance of the strategies can be observed across different $\alpha$ values:

In the Enron and Lucene datasets, the rank of performance is V0 followed by V1, and then V2. In the Wikipedia dataset, the rank of performance is V2 followed by V0, and then V1.

This pattern aligns with the observations made in Figure 4.5, but there is a notable difference in the slopes of the curves. The more gentle slopes seen in Figure 4.6 are attributed to the impact of the $\alpha$ values used in the attack. A larger $\alpha$ value leads to more accurate document identification, resulting in a steady and gradual increase in the document recovery rate.

An interesting observation in the Wikipedia dataset is that all three strategies seem to converge to the same document recovery rate when the $\alpha$ value is set to 10.




**(a)** Enron **(b)** Lucene **(c)** Wiki

**Figure 4.6:** Document Recovery Performance ($\alpha$)

Next, we aim to compare the document recovery results of our attack with an existing attack that has achieved the best record in document recovery, which is the VAL attack.

From the perspective of the overall variation tendency, comparing the document recovery results of our attack (Figure 4.5) with the results of the VAL attack (Figure 4.7), the VAL attack shows an even more stable performance, the recovery rates trends to remain at a stable level from leakage of 0.5%. In contrast, our attack requires a higher data leakage level to reach a similar stable status, specifically at 1% for V0 and V1, and 5% for V2 in the Enron and Lucene datasets.

When comparing the concrete document recovery rates, our attack achieves a lower document recovery rate for the Enron dataset, a similar recovery rate for the Lucene dataset, and significantly better results for the Wikipedia dataset.

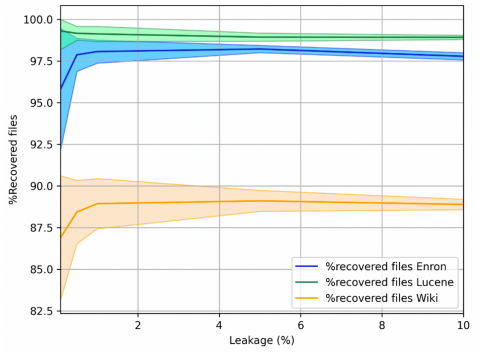



**Figure 4.7:** Document Recovery of VAL [24]

In summary, our attack consistently achieves document recovery rates higher than 90% with a data leakage level of 5%. Moreover, we can reach this 90% document recovery rate even before the $\alpha$ value reaches 10. These results demonstrate that our approach of recovering documents by selecting $\alpha$ keywords from each document is both feasible and effective, yielding satisfactory document recovery outcomes.

Furthermore, when comparing our results with the outcomes of the VAL attack, which is considered a state-of-the-art inference attack, we find that our attack produces comparable document recovery rates. This indicates that our proposed attack is competitive and capable of achieving document recovery rates similar to those obtained by the VAL attack.

Query Recovery:
In the analysis of query recovery, we aim to verify whether our attack has achieved the proposed amplifier effect. To illustrate this, we present two tables, namely Table 4.8a and Table 4.8b.

Table 4.8a displays the specific number of keywords that have been injected and subsequently recovered by the attack. It provides a clear indication of the total count of injected and recovered

keywords. The numbers are decimal values since they represent the average from all experimental trials.

In Table 4.8b, we use the criterion ”rate of return” to visualize the average number of keywords that can be recovered by each injected keyword. This metric quantifies the effectiveness of our attack in terms of keyword recovery.

In Table 4.8a, we observe that both the number of injected and recovered keywords increase with an increase in the data leakage level or the value of $\alpha$. This behavior is natural since the size of the leaked data indicates how many documents we can choose keywords from, and the value of $\alpha$ determines how many keywords we should select from each document.

The noteworthy difference in this table lies in the numbers of injected under different strategies.

Comparing V0 and V1, we can observe no significant difference in the number of injected keywords. However, V2 consistently achieves the largest amount of injected keywords. This can be attributed to V2's strategy of choosing the global least frequent keywords, which reduces the likelihood of redundancy among the selected keywords.

Regarding the number of recovered queries, we can observe a consistent pattern where the quantity is always larger than the number of initially injected keywords in this table. However, to precisely analyze the specific extent of this increase, we rely on the ”rate of return” metric.

In Table 4.8b, we can observe that all the obtained rates of return are greater than 1. This confirms that more than one query can be recovered with each injected keyword, demonstrating the success of our proposed attack in achieving the expected amplifier effect.

To facilitate a comparison between strategies, we label the values in color. In this color scheme, blue indicates that the rate of return in the current setting is lower than 1.5, while red indicates a value that exceeds 3. The darkness of the color corresponds to the proximity of the value to the extremes.

We can observe that V1 has the most red-colored labels, indicating that it has the highest rate of return and the most effective amplifier effect. V0 is the next best, with a few red labels and some moderate blue labels. V2, however, has no red labels and the darkest blue label, indicating that it has the weakest amplifier effect among the three strategies.

The accuracy of query recovery, however, surprised us with unexpectedly low recovery rates, as shown in Figure 4.10 and 4.11.

In the previous analysis of the document recovery results, we observed that although our attack did not achieve exactly the same recovery rates as the VAL attack, the results were still comparable. Therefore, we were expecting a similar outcome in query recovery (Figure 4.9). However, the query recovery rates we obtained were significantly lower than expected. For instance, under the data leakage of 10%, the query recovery rate could hardly exceed 55% in the Enron and Lucene datasets, and it was even lower in the Wikipedia dataset, where the accuracy did not exceed 50%. In contrast, the VAL attack achieved a nearly 100% query recovery rate.

The document recovery results have remained at this expected level even under different values of $\alpha$ (Figure 4.11). The highest document recovery rate we achieved was slightly above 60% in the Lucene dataset with a $\alpha$ value of 10. The overall trend of accuracy still shows a slight increase as we increase the value of $\alpha$. Nevertheless, this increase in accuracy is primarily attributed to the higher number of injected keywords rather than the attack's potential for high accuracy when $\alpha$ is greater than 10.

## 4.5. Discussion

In summary, our proposed attack demonstrates the capability to achieve satisfactory document recovery rates with all three strategies. Moreover, the document recovery rate tends to increase as we increase the $\alpha$ value or the size of data leakage until it reaches a stable state. For instance, in the Lucene dataset, strategies V0 and V1 become stable at around 98% of the document recovery rate when $\alpha$ reaches 5 or when data leakage reaches 1%.

Furthermore, by looking at the rate of return, we confirmed the ability of the amplifier effect of this attack as we designed. Additionally, if we want to achieve a stronger effect, V1 might be the most optimal option among all three strategies.

| Dataset Knowledge | | | | Injected Keyword / Recovered Keyword | | |
|---|---|---|---|---|---|---|
| Dataset | Leaked Doc. (%) | No. of KW | Alpha | V0 | V1 | V2 |
| Enron | 0.1 | | 2 | 28.2 / 37.8 | 28.0 / 35.8 | 29.8 / 47.2 |
| | | | 3 | 43.6 / 77.4 | 41.8 / 66.4 | 44.6 / 74.4 |
| | | | 5 | 68.2 / 100.6 | 68.6 / 94.6 | 72.6 / 106.0 |
| | | | 10 | 125.2 / 154.8 | 125.4/148.6 | 138.2 / 167.8 |
| | 0.5 | | 2 | 134.0 / 467.2 | 120.6 / 380.4 | 146.8 / 311.6 |
| | | | 3 | 194.6 / 483.6 | 177.4 / 449.4 | 217.4 / 421.8 |
| | | | 5 | 294.6 / 563.8 | 282.8 / 555.8 | 355.0 / 588.2 |
| | | | 10 | 507.4 / 726.6 | 496.4 / 707.0 | 660.0 / 902.0 |
| | 1 | | 2 | 255.2 / 947.8 | 221.6 / 873.4 | 293.2 / 771.4 |
| | | | 3 | 358.4 / 1032.8 | 323.2 / 977.8 | 433.8 / 1006.8 |
| | | | 5 | 534.0 / 1165.0 | 508.0 / 1137.8 | 701.4 / 1292.8 |
| | | | 10 | 901.2 / 1388.0 | 880.8 / 1375.0 | 1282.8 / 1846.4 |
| | 5 | | 2 | 909.4 / 2874.6 | 779.2 / 2778.0 | 1383.8 / 3002.0 |
| | | | 3 | 1194.6 / 3073.2 | 1084.0 / 2992.8 | 2000.4 / 3587.6 |
| | | | 5 | 1704.8 / 3382.6 | 1568.2 / 3299.8 | 3085.0 / 4493.8 |
| | | | 10 | 2595.4 / 3885.0 | 2501.6 / 3838.8 | 4942.2 / 5767.8 |
| | 10 | | 2 | 1495.6 / 4656.6 | 1285.8 / 4466.0 | 2626.0 / 5047.8 |
| | | | 3 | 1936.2 / 4919.0 | 1739.4 / 4783.8 | 3724.2 / 5986.4 |
| | | | 5 | 2650.0 / 5293.8 | 2471.8/ 5228.6 | 5516.6 / 7300.8 |
| | | | 10 | 3924.2 / 5968.8 | 3824.4/5958.8 | 8145.6 / 9030.2 |

**(a)** Recovered Queries

< 1.5
> 3.0

| Dataset Knowledge | | | | Rate of return | | |
|---|---|---|---|---|---|---|
| Dataset | Leaked Doc. (%) | No. of KW | Alpha | V0 | V1 | V2 |
| Enron | 0.1 | | 2 | 1.340 | 1.279 | 1.584 |
| | | | 3 | 1.775 | 1.589 | 1.668 |
| | | | 5 | 1.475 | 1.379 | 1.460 |
| | | | 10 | 1.236 | 1.185 | 1.214 |
| | 0.5 | | 2 | 3.487 | 3,154 | 2.123 |
| | | | 3 | 2.485 | 2.533 | 1.940 |
| | | | 5 | 1.914 | 1.965 | 1.657 |
| | | | 10 | 1.432 | 1.424 | 1.367 |
| | 1 | | 2 | 3.714 | 3.941 | 2.631 |
| | | | 3 | 2.882 | 3.025 | 2.321 |
| | | | 5 | 2.1820 | 2.240 | 1.843 |
| | | | 10 | 1.540 | 1.561 | 1.439 |
| | 5 | | 2 | 3.161 | 3.565 | 2.169 |
| | | | 3 | 2.573 | 2.761 | 1.793 |
| | | | 5 | 1.984 | 2.104 | 1.457 |
| | | | 10 | 1.497 | 1.535 | 1.167 |
| | 10 | | 2 | 3.114 | 3.473 | 1.922 |
| | | | 3 | 2.541 | 2.750 | 1.607 |
| | | | 5 | 1.998 | 2.115 | 1.323 |
| | | | 10 | 1.521 | 1.558 | 1.109 |

**(b)** Rate of Return

**Figure 4.8:** Query Recovery Performance

However, the query recovery results we have observed are disappointing, falling far below our expectations.

In our effort to analyze the disappointing query recovery results, we first thoroughly examined our code implementation and algorithm logic, but we did not find any issues that could explain the low performance. Consequently, we shifted our focus to explore another potential factor: the difference in the keyword universe used in our attack compared to the VAL attack.

The Subgraph attack, as mentioned in Chapter 3, provided some insights into the relationship between the frequency of keywords in the keyword universe and the results of query recovery.

This naturally leads us to consider whether the use of a different keyword universe might be the true reason behind the low query recovery rate we observed. Unlike the VAL attack, which used the most frequent 5000 keywords as the keyword universe, we chose to use all extracted keywords to form the keyword universe in our experiments.

In the next chapter, we plan to verify the insight about the impact of the keyword universe on query

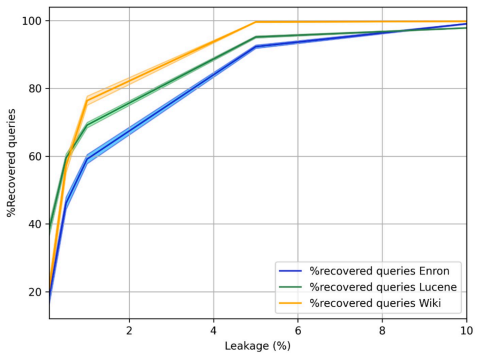
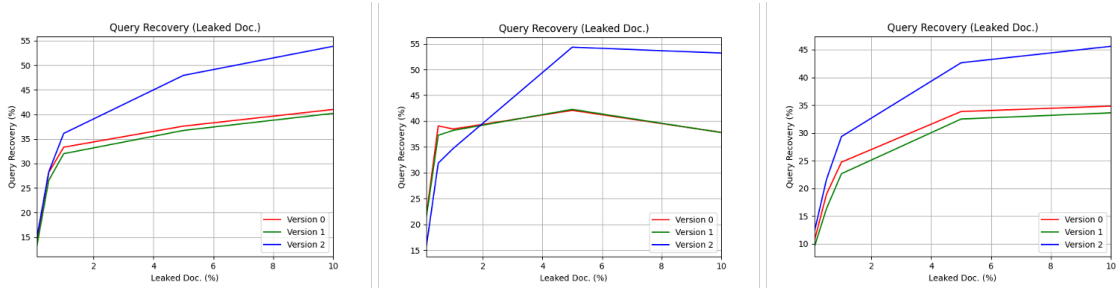
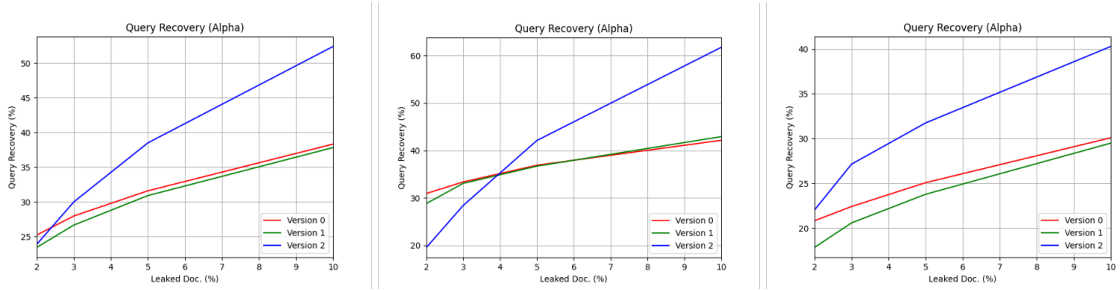



**Figure 4.9:** Query Recovery of VAL [24]




**(a)** Enron **(b)** Lucene **(c)** Wiki

**Figure 4.10:** Query Recovery Performance (Leaked Doc%)




**(a)** Enron **(b)** Lucene **(c)** Wiki

**Figure 4.11:** Document Recovery Performance ($\alpha$)

recovery results. We will do a reproduction of the VAL attack, but this time we will replace its keyword universe with the one we used in our proposed attack. Furthermore, we will investigate the reasons behind the better query recovery results achieved by the high-frequency keyword set.

# 5 Discovery

In this chapter, our primary focus is to investigate and understand the reasons behind the significant disparity between the query recovery results obtained in our initial attempt and the expected results proposed in the VAL attack.

To begin our investigation, we plan to verify whether the low query recovery rate we obtained in the experiment was indeed caused by the use of a wider keyword universe. We will conduct a reproduction of the VAL attack, but this time, we will replace its original keyword universe with the one we used in our proposed attack.

## 5.1. Reproduction of VAL

### 5.1.1. Setup

To ensure a fair comparison and eliminate any potential coding-related issues, we will use the provided code in [24] for the reproduction of the VAL attack.

We will use the Enron dataset and apply the same preprocessing steps, such as removing English stopwords and stemming the remaining keywords, to create a consistent experimental setup.

The experiment will run with the same set of data leakage levels: 0.1%, 0.5%, 1%, 5%, and 10%. Also, for making the comparison, this reproduction is conducted with two different keyword universes. The first keyword universe will consist of the top 5000 frequent keywords, the same as what the VAL attack used. The second keyword universe will include all keywords extracted from dataset $W$, the same as what we used in our proposed attack.

Each experiment will run 10 times.

### 5.1.2. Result

The results obtained with the same keyword universe used by VAL, as depicted by the red line in Figure 5.1, confirm the document and query recovery rates proposed by the VAL attack. This indicates that the code we used ran well and reproduced the expected results.

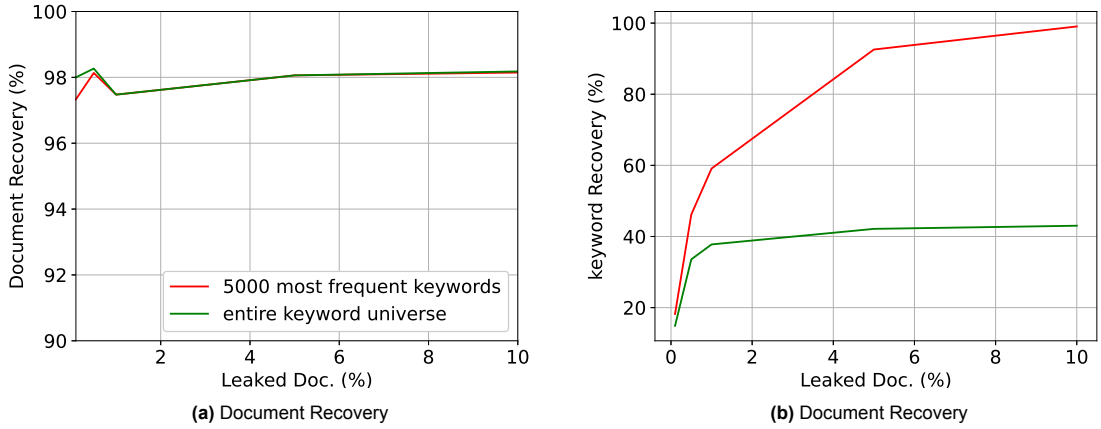
Specifically, the document recovery in the reproduction achieves a consistent accuracy of 98%, and the query recovery accuracy exceeds 90% even in the presence of a 5% data leakage. Moreover, as the data leakage increased to 10%, the accuracy of query recovery approached 100%.

However, when we examine the green line in the figure, we observe notable deviations in the results of query recovery. The accuracy of query recovery with 5% leakage dropped from above 90% to around 42%, with no significant increase observed when the leaked data increased to 10%.

Additionally, by using all extracted keywords as the keyword universe instead of only the top 5000 most frequent keywords, a significant increase in running time was observed.

### 5.1.3. Conclusion

The results we obtained from this reproduction align with our assumption about the reason behind the unexpectedly low query recovery in the previous chapter. The use of a wider keyword universe leads to a significant decrease in the accuracy of query recovery.




**(a)** Document Recovery **(b)** Document Recovery

**Figure 5.1:** Results of VAL reproduction

The reason behind this situation might not solely be the frequency of keywords itself, but rather the unique occurrence patterns associated with high-frequency keyword sets.

Our intuition leads us to believe that high-frequency keyword sets might have a larger proportion of unique occurrence patterns, which is a key factor for inference attacks to identify keywords.

To confirm this hypothesis, we conducted an analysis over all keywords in the Enron dataset and counted how many of them own unique occurrence patterns among all documents. The findings of this analysis are presented in the following section.

## 5.2. Keyword classification

For better presenting the result of our analysis, we now introduce a new classification of keywords into two categories: inert keywords and active keywords. These terms are inspired by the chemical terms ”inert gas” and ”active gas,” respectively. Active keywords possess a unique access pattern and can be effectively identified from all the keywords, whereas inert keywords lack a unique access pattern.

The reason behind proposing this classification is that there was no such terminology used to indicate keywords with or without unique occurrence patterns. In previous research, keywords were typically grouped only by their occurrence frequencies, and alternative classifications of keywords received limited attention. As a result, we propose the distinction between inert and active keywords based on their occurrence patterns to address this vacancy.

We display the result of the analysis in Figure 5.2.

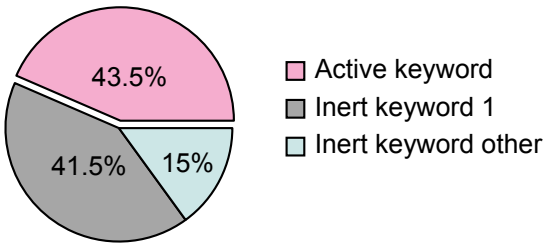



**Figure 5.2:** Keyword distribution (Enron)

**Active keywords**, labeled by the pink color, represent the proportion of keywords with unique access patterns in each dataset.

**Inert keywords 1**, labeled by dark gray, represent keywords that appeared in only one document. It is interesting to note that these keywords also belong to the group of least frequent keywords since they appeared only once in the entire dataset.

**Inert keywords other**, labeled by light blue, indicate keywords that appeared in more than one document.

The pie chart reveals a surprising observation that the proportion of active keywords in the dataset is much lower than we initially imagined. Out of the total 63,029 keywords in the dataset, only 27,444 were identified as active keywords. This accounts for approximately 43.5% of the total keywords.

This finding aligns with the query accuracy obtained during the replication of the VAL attack, as depicted by the green line in Figure 5.1b, which also approximates the 43% mark. The low proportion of active keywords is likely a significant factor contributing to the reduced query recovery accuracy.

The selection of the 5000 most common keywords used in the VAL attack reveals a high proportion of active keywords: our classification approach identified 4996 out of the 5000 keywords as active keywords, resulting in an almost 100% within this subset.

As a result, the experimental results reported in VAL achieved high accuracy due to the dominance of active keywords within this subset. The presence of a significant number of active keywords with unique access patterns contributed to their successful query recovery.

### 5.2.1. Discussion

The proposed keyword classification, along with the comparison results, not only provides an explanation for the unexpected query recovery rate but also highlights the significant challenge of achieving a high query recovery accuracy for inference attacks when operating within a larger keyword universe. The presence of a high proportion of inert keywords in the keyword universe can lead to a low query recovery rate, even in the absence of any countermeasures.

To overcome this obstacle, inference attacks must either discover alternative leakage patterns capable of distinguishing between two queries with identical appearances in documents or leverage the assistance of other SSE attacks, such as file injection attacks, which can generate unique patterns for target keywords.

It is intriguing to observe that our proposed attack, initially designed to enhance the performance of file injection attacks using inference attack concepts, has found an additional purpose: assisting inference attacks in overcoming the dilemma caused by inert keywords.

This unexpected discovery highlights the adaptability and necessity of our attack. By combining the strengths of inference attacks and file injection attacks, our proposed approach offers a promising solution to address the challenge of achieving higher query recovery rates in scenarios with a larger keyword universe (for inference attacks), while reducing the number of injected keywords needed for successful recovery (for file injection attacks).

In the next chapter, we would like to show an updated version of our proposed attack to better reach these goals.

# 6

# Improved Design

In this chapter, we presented an updated design for our proposed attack, incorporating the idea of the newly proposed keyword classification.

## 6.1. Notation

Similar to the initial trial, we begin this chapter by presenting a table of notation. Table 6.1 provides a list of the notations used in our proposed attack and their corresponding explanations.

As in the initial scenario, we consider a situation with $n$ server documents $ED$ and $n'$ leaked files $D'$. The size of the keyword universe $W$ and the corresponding query universe $Q$ is $m$, while $m'$ denotes the size of keywords that can be extracted from $D'$ as $W'$.

The difference from Table 4.1 is the addition of $W'_{ac}$, which represents the keywords in $W'$ that possess a local unique access pattern.

**Table 6.1:** Summary of notations used in our attack.

| Notations | Definition |
|---|---|
| $D$ | Plaintext document set, $D = \{d_1, ..., d_n\}$ |
| $ED$ | Server document set, $ED = \{ed_1, ..., ed_n\}$ |
| $D'$ | Leaked document set, $D = \{d_1, ..., d_{n'}\}$ |
| $W$ | Keyword universe, $W = \{k_1, ..., k_m\}$ |
| $Q$ | Query set, $Q = \{q_1, ..., q_m\}$ |
| $W'$ | Known Keyword set, $W' = \{k_1, ..., k_{m'}\}, W' \in W$ |
| $W'_{ac}$ | Active keyword set in known keywords, $W'_{ac} = \{k_1, ..., k_{m'_{ac}}\}, W'_{ac} \in W'$ |
| $W'_i$ | Chosen keyword set, $W' = \{k_1, ..., k_{m'_i}\}, W'_i \in W'$ |
| $IF$ | Inject file set, $IF = \{id_1, ..., id_k\}$ |
| $EIF$ | Encrypted injected file set, $EIF = \{eid_1, ..., eid_k\}$ |
| $T$ | Limitation of the number of keywords in each document |
| $\|d_i\|$ | Number of keywords/queries in $d_i$ |
| $\|D\|$ | Number of files in $D$ |
| $\alpha$ | Number of keywords selected from each document |
| $C$ | Set of matched documents |
| $C_i$ | Set of matched injected documents |
| $R$ | Set of matched queries |
| $R_i$ | Set of matched documents by file injection |

## 6.2. The Design

As we have come to understand that the query recovery rate is not directly related to the frequency of keywords, but rather to how many of them have a unique access pattern, we have decided to update the

way our proposed attack chooses injected keywords. Instead of continuing to select injected keywords solely from low-frequency keywords, we will now choose them specifically from inert keywords.

The main reasons for making this update are based on the following two considerations:

**Inert Keywords as Low-Frequency Keywords:** In our previous attempts, we successfully used either randomly chosen or low-frequency keywords to identify leaked documents, which yielded relatively satisfactory results in document recovery. This finding demonstrated the feasibility of using low-frequency keywords for document recovery.

From the pie chart of Enron data structure (Figure 5.2), we observe that approximately (41.5%+15% =) 56.5% of the keywords are inert keywords, out of which ($\frac{41.5\%}{56.5\%}$=) 73.3% occur only once in the entire dataset. This significant proportion of one-time occurrences represents a large number of low-frequency keywords, making them the least frequent keywords, both locally and globally, that can be used for document identification.

**Minimized Overlap Between Injected and Recovered Keywords:** Choosing inert keywords as injected keywords has an additional advantage. As the subsequent inference process can only recover active keywords, employing inert keywords as injected keywords ensures minimal overlap between the injected and recovered keywords. This reduction in overlap maximizes the final amplifier effect of the attack and enhances its overall efficiency.

With the current approach to keyword selection, we do not foresee a significant reduction in the size of injected documents. Nevertheless, we expect to achieve a more robust overall rate of return in query recovery, leading to an enhanced amplifier effect. Simultaneously, our objective is to ensure that the final recovered keywords exceed the proportion of active keywords, even when utilizing the entire keyword universe of the leaked documents $D$.

### 6.2.1. Leaked Knowledge

In this updated attack, we continue to assume that the attacker possesses knowledge of a partial set of leaked data and the corresponding access patterns:

The attacker is aware of a set of leaked documents $D' = d_1, ..., d_{n'}$, the set of known keywords $W' = k_1, ..., k_{m'}$, the set of encrypted documents $ED = ed_1, ..., ed_n$, and the corresponding set of queries $Q = q_1, ..., q_m$, where $n' \leq n$ and $m' \leq m$. Additionally, the attacker possesses information about the number of keywords in the leaked documents, denoted by $|d_i|$, and the number of queries in the encrypted documents, denoted by $|ed_j|$.

## 6.3. Procedure

The updated attack retains the same structure as the previous version, consisting of three key procedures: keyword selection, file injection, and recovery. The concrete steps and algorithms for file injection and recovery procedures even remain unchanged, so in this section, we will provide a detailed explanation of the changes made to the keyword selection procedure.

The key change involves an additional step called ”Keyword Classification,” which aims to group the keywords based on their access patterns before conducting the keyword selection process. The updated attack model is depicted in Figure 6.1, illustrating the insertion of the keyword classification step before the selection of $W'_i$.

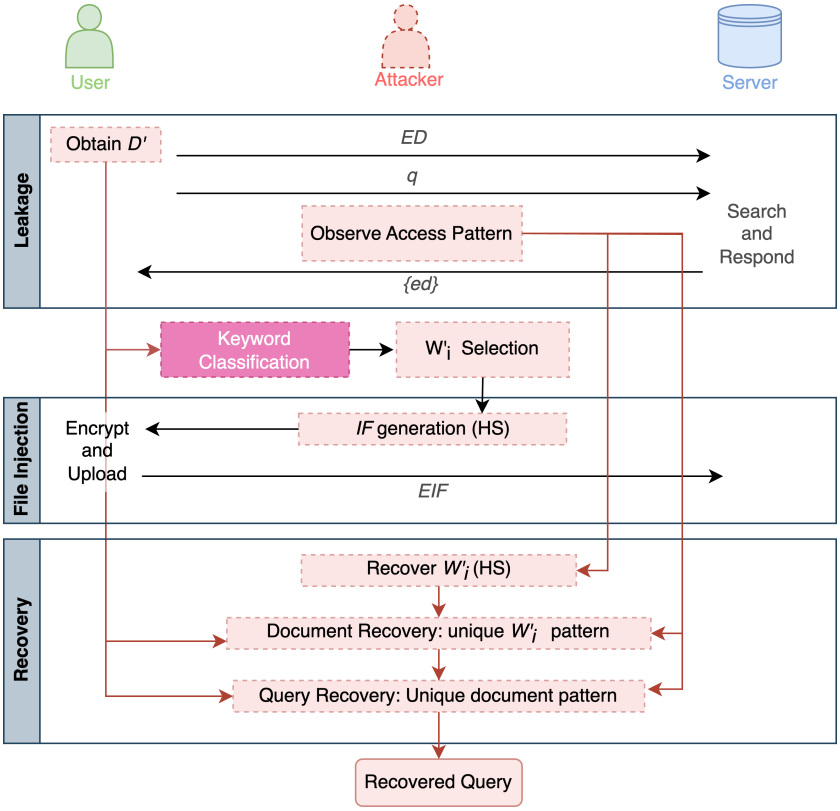



**Figure 6.1:** Attack model

### Updated Keywords Selection

After obtaining the leaked documents $D'$, the attacker proceeds to extract the corresponding keyword set $W'$. At the beginning of the keyword selection algorithm, the nested algorithm Algorithm 7 is executed to categorize $W'$ into two distinct groups: active keywords, denoted as $W'ac$, and local inert keywords, that $k_i \in W'$ and $k_i \notin W'_{ac}$.

It is important to note that the active keywords that exhibit unique patterns in $D'$ also imply that they are active keywords in the entire dataset, regardless of locality. In other words, there is no difference between local or global active keywords.

However, keywords that do not display uniqueness within $D'$ may still potentially exhibit unique patterns in documents that were not leaked. Hence, these inert keywords found by the attacker in the leaked documents are referred to as local inert keywords.

After obtaining the set of active keywords $W'_{ac}$, the attacker proceeds to select the keywords for injection using Algorithm 8:

For each document, $\alpha$ local inert keywords are selected to represent the document.

If the number of local inert keywords in a document is less than $\alpha$, the remaining keywords are chosen from $W'_{ac}$ based on their lowest frequency in the document.

The algorithm returns a set of selected keywords $W'_i$, where the maximum size of $W'_i$ is $\alpha \cdot n'$.

After keyword selection $W'_i$ will serve as the input for the file injection procedure, similar to what we explained in Chapter 4.

**Algorithm 7** Keyword_Classification

**Input**: A set of known document $D'$ and corresponding set of known keyword $W'$
**Output**: A set of inert keyword $W'_{ac}$

**procedure** Keyword_Classification($D'$, $W'$)
    Initialize an empty dictionary $Combination$ and a empty list $Count$
    **for** $i = 1 \rightarrow$ #$W'$ **do**
        $com \leftarrow [w_i$ in $d_j$ where $d_j \in D']$
        **if** $com$ not in $Combination$ **then**
            $Count$.append(1)
            $Combination[len(Count) - 1]['com'] \leftarrow com$
            $Combination[len(Count) - 1]['keyword'] \leftarrow w_i$
        **else**
            $index \leftarrow$ key of $com$ in $Combination$
            $Count[index] \leftarrow Count[index] + 1$
        **end if**
    **end for**
    Initialize an empty list $W'_{ac}$
    **for** $i = 1 \rightarrow$ #$Count$ **do**
        **if** $Count[i] == 1$ **then**
            $W'_{ac}$.append($Combination[i]['keyword']$)
        **end if**
    **end for**
    **return** $W'_{ac}$
**end procedure**

**Algorithm 8** Keyword_Selection

**Input**: A set of known document $D'$ and corresponding set of known keyword $W'$, and a variable $\alpha$
**Output**: A set of chosen keyword for injection $W'_i$

**procedure** Keyword_Selection($D'$, $\alpha$)
    $W'_{ac} \leftarrow$ Keyword_Classification$(D', W')$ ▷ Return with the set of local active keyword in $D'$
    Initialize an empty list $W'_i$
    **for** $i = 1 \rightarrow$ #$D'$ **do**
        $k \leftarrow \alpha$ keywords in $d_i$, that keywords $\notin W'_{ac}$
        **if** $|k| < \alpha$ **then**
            add $\alpha - |k|$ least frequent keywords in $d_i$, and keywords $\in W'_{ac}$ to $k$
        **end if**
        $W'_i \leftarrow W'_i \cup k$ ▷ excluding the repetition
    **end for**
    **return** $W'_i$
**end procedure**

# 7 Experiment

In this chapter, we will present the experimental results of the updated version of our proposed attack.

## 7.1. Setup

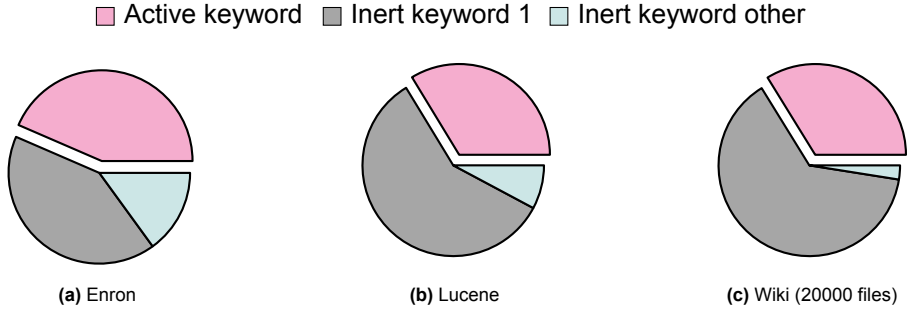
### 7.1.1. Datasets

Continuing with the same approach as in our previous experiments, we will utilize three real-life datasets, namely Enron [26], Lucene [14], and Wikipedia, to evaluate the performance of the proposed attack.

To preprocess the datasets, we removed common English stopwords and stemmed the keywords to their root forms.

### 7.1.2. Distribution of Dataset

What sets this experiment apart from the previous one is that we performed an analysis of the keyword distribution in the datasets to identify the proportions of inert and active keywords in each dataset. This analysis enables us to compare the results with our final query recovery rates and evaluate whether we have achieved our objective of recovering more than just active keywords. The distribution of keywords in the datasets is visually depicted in Figure 7.1.




**(a)** Enron **(b)** Lucene **(c)** Wiki (20000 files)

**Figure 7.1:** Keyword Distribution of Datasets

We observed a similar proportion of active keywords in Lucene and Wikipedia: Lucene has 33.7% active keywords, while Wikipedia has 33.8%. Additionally, 66.3% of the keyword universe in Lucene consists of inert keywords, and this proportion is 66.2% in Wikipedia.

However, there was a notable difference between Lucene and Wikipedia in terms of the proportion of inert keywords that appeared in more than one document. Lucene had a higher percentage (7.8%) of inert keywords that occurred in multiple documents, while Wikipedia had a lower percentage (2.5%).

On the other hand, Enron exhibited a higher proportion of active keywords, with 43.5% of the keyword universe having a unique access pattern. Furthermore, 41.5% of the keywords in Enron were inert keywords that appeared only once in the dataset, and 15% of the keywords were inert keywords that appeared in more than once.

### 7.1.3. Other Setup

In contrast to the previous trial, for this experiment, we utilized the entire datasets.

To assess the effectiveness of the attack, we conducted evaluations with varying percentages of leaked dataset, including 0.1%, 0.5%, 1%, 5%, and 10%. Additionally, we explored different values of $\alpha$ (2, 3, 5, and 10), and each experiment was repeated 10 times.

## 7.2. Result and Comparison

In both the document and query recovery sections, we will begin by analyzing the results by comparing their performance across different datasets and also compare with our first experiment. Subsequently, we will compare the outcomes of our attack with the document and query recovery results of LEAP and VAL attacks, specifically using the Enron dataset.

The reason behind conducting the comparison experiments exclusively with only one dataset is to consider the running time. Since we utilize a larger keyword universe for all attacks, LEAP and VAL are expected to experience longer running times than they did in their original experiments for each round. Among all three datasets, Enron has the least number of keywords.

Moreover, since we do not need to compare different strategies this time, the lines in different colors will represent the results obtained with different $\alpha$ values.

### 7.2.1. Document Recovery

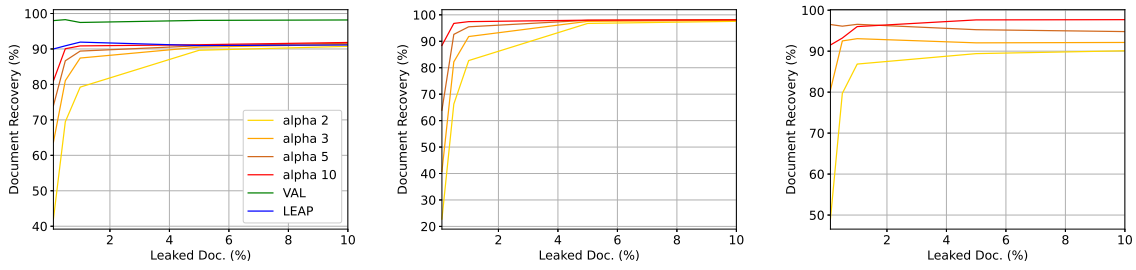
Figure 4 shows that $\alpha$ values of 3, 5, and 10 achieve similar accuracy levels after a 1% data leakage for the Enron and Lucene datasets. In contrast, $\alpha$ 2 reaches a comparable accuracy after a 5% leakage. This suggests that for data leakages larger than 5% in Enron and Lucene, using $\alpha$ 2 is sufficient to obtain satisfactory document recovery results. It is worth noting that Lucene exhibits higher overall accuracy in document recovery, likely due to a larger proportion of inert keywords in its dataset.

In the case of the Wikipedia dataset, higher $\alpha$ values consistently yield better performance in document recovery. This indicates that inert keywords that only appear in a single document have less power in identifying and recovering documents since they can only indicate the presence of a specific document.

If we recall the document recovery results from the first experiment in Figure 4.5, we can confirm the ability of the updated attack in document recovery, as it achieves a comparable recovery rate.

**Comparing with LEAP and VAL:**

All $\alpha$ values demonstrate similar document recovery accuracy to LEAP, which is lower than the accuracy of VAL that leverages an extra volume pattern. This indicates that our attack, which leverages only the access pattern, can achieve satisfactory document recovery comparable to that proposed by LEAP with a single-round match.




**(a)** Enron **(b)** Lucene **(c)** Wikipedia

**Figure 7.2:** Document Recovery performance

### 7.2.2. Query Recovery

In Table 7.1, we present the concrete number of injected and recovered keywords. The table shows the number of injected and recovered keywords under different settings of data leakage and $\alpha$ value for

three datasets: Enron, Lucene, and Wikipedia. Additionally, we provide the size of known keywords $W'$ under different data leakage scenarios within each dataset as a reference to evaluate the performance of query recovery.

Based on the data presented in this table, we can make a preliminary conclusion about the effectiveness of the amplifier effect achieved by the updated attack. However, for a more detailed evaluation of the extent of the amplifier effect, we need to consider the table of the rate of return (Table 7.2).

**Table 7.1:** Number of Recovered Query

| **Dataset Knowledge** | | **Injected Keyword / Recovered Keyword (On Average)** | | |
|---|---|---|---|---|
| **Leaked Doc. (%)** | **Alpha** | **Enron** | **Lucene** | **Wiki** |
| 0.1 | 2 | 58.1 / 80.4 | 99.4 / 108.1 | 39.3 / 51.8 |
| | 3 | 86.1 / 138.3 | 148.7 / 180.5 | 58.5 / 110.1 |
| | 5 | 139.0 / 224.2 | 247.1 / 342.6 | 96.8 / 178.0 |
| | 10 | 257.3 / 350.4 | 474.7/ 732.9 | 181.2 / 261.4 |
| **No. of KW** | | 1137.1 | 1646.8 | 1425.2 |
| 0.5 | 2 | 286.6/ 644.5 | 493.6 / 814.5 | 195.1 / 706.9 |
| | 3 | 422.2 / 913.4 | 729.3 / 1319.7 | 290.8 / 1033.3 |
| | 5 | 671.0 / 1255.1 | 1163.4 / 2110.0 | 471.0 / 1260.3 |
| | 10 | 1168.9/ 1773.9 | 1978.0 / 3076.4 | 864.6 / 1596.9 |
| **No. of KW** | | 3581.8 | 4295.5 | 3973.3 |
| 1 | 2 | 548.8 / 1398.8 | 949.1 / 1843.7 | 380.3 / 1764.4 |
| | 3 | 804.8 / 1856.8 | 1389.9 / 2736.8 | 570.5 / 2184.7 |
| | 5 | 1248.3 / 2337.0 | 2141.6 / 3746.0 | 925.2 / 2558.5 |
| | 10 | 2067.6 / 3059.5 | 3423.5 / 5046.0 | 1683.0 / 3170.0 |
| **No. of KW** | | 5027.7 | 6686.5 | 6387.7 |
| 5 | 2 | 2209.0 / 5133.5 | 3746.9 / 8455.2 | 1739.5 / 7248.6 |
| | 3 | 3092.2 / 5990.6 | 5180.7 / 9942.6 | 2566.6 / 8150.2 |
| | 5 | 4443.5 / 7128.0 | 7274.1 / 11748.2 | 4038.4 / 9388.9 |
| | 10 | 6527.8 / 8720.5 | 10115.2 / 13786.0 | 6726.0 / 11241.8 |
| **No. of KW** | | 12740.3 | 19877.6 | 19255.2 |
| 10 | 2 | 3705.5 / 8105.7 | 6108.8 / 13630.8 | 3250.7 / 12403.0 |
| | 3 | 5079.6 / 9335.6 | 8229.0/ 15502.6 | 4739.0 / 13790.7 |
| | 5 | 7102.1 / 10993.0 | 11192.5 / 17873.1 | 7268.3 / 15751.1 |
| | 10 | 10063.2 / 13188.1 | 15148.2 / 20506.0 | 11592.7 / 18616.5 |
| **No. of KW** | | 18478.4 | 27733.4 | 31995.1 |

### Rate of Return

The rate of return tends to show interesting trends under different data leakage levels and $\alpha$ values. For instance, under a low data leakage level, a higher $\alpha$ value results in a greater rate of return, as observed in Lucene. At a data leakage level of 0.1%, the rate of return for $\alpha$ 2, 3, 5, and 10 is 1.088, 1.214, 1.386, and 1.544, respectively, displaying an increasing trend. However, as the data leakage level increases to 10%, the rate of return for these $\alpha$ values becomes 2.231, 1.884, 1.597, and 1.354, indicating a decreasing trend. Among all data leakage levels, choosing the middle value of $\alpha$ 3 or 5 ensures a more stable rate of return.

Once a 10% data leakage level is reached, the rate of return tends to stabilize, with a smaller difference in the rate of return between $\alpha$ 2 and 3. Both the Enron and Lucene datasets exhibit similar rate of return values, with the lowest being 1.311 in the Enron dataset.

Among the three datasets, the proposed attack is more effective for Wikipedia, which demonstrates an overall higher rate of return. This suggests that the attack performs better in datasets with a larger proportion of inert keywords, particularly those that occur only once in the dataset.

**Comparing with File Injection Attacks**

At this point, we can make a comprehensive comparison between our attack and two existing file injection attacks based on access patterns: Hierarchical Search attack [44], and Decoding [3].

**Table 7.2:** Rate of return

| Dataset Knowledge | | Rate of Return (On Average) | | |
|---|---|---|---|---|
| **Leaked Doc. (%)** | **Alpha** | **Enron** | **Lucene** | **Wiki** |
| 0.1 | 2 | 1.384 | 1.088 | 1.318 |
| | 3 | 1.606 | 1.214 | 1.882 |
| | 5 | 1.613 | 1.386 | 1.839 |
| | 10 | 1.362 | 1.544 | 1.443 |
| 0.5 | 2 | 2.249 | 1.650 | 3.623 |
| | 3 | 2.163 | 1.810 | 3.553 |
| | 5 | 1.870 | 1.814 | 2.676 |
| | 10 | 1.518 | 1.555 | 1.847 |
| 1 | 2 | 2.549 | 1.943 | 4.639 |
| | 3 | 2.307 | 1.969 | 3.829 |
| | 5 | 1.872 | 1.749 | 2.765 |
| | 10 | 1.480 | 1.474 | 1.884 |
| 5 | 2 | 2.324 | 2.257 | 4.167 |
| | 3 | 1.937 | 1.919 | 3.175 |
| | 5 | 1.604 | 1.615 | 2.325 |
| | 10 | 1.336 | 1.363 | 1.671 |
| 10 | 2 | 2.187 | 2.231 | 3.815 |
| | 3 | 1.838 | 1.884 | 2.910 |
| | 5 | 1.548 | 1.597 | 2.167 |
| | 10 | 1.311 | 1.354 | 1.606 |

For both Hierarchical Search attack and Decoding, the highest rate of return is 1, indicating that all injected keywords lead to correct query recovery. On the other hand, our attack, as shown in Table 7.2, consistently achieves a rate of return greater than 1, even in the worst-case scenarios. For instance, in the Lucene dataset with an $\alpha$ value of 2, the rate of return is 1.088.

Once the data leakage level exceeds 10%, our attack demonstrates stability, and we observe that a single keyword injection can recover at least 1.3 queries. This finding suggests that in order to correctly recover the same number of queries $|R|$, we only need to inject with a maximum injected keyword size of $W_i' = \frac{|R|}{1.3}$.

Moreover, the number of injected files can be calculated as $|IF| = \lceil \frac{\frac{|R|}{1.3}}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$. Comparing this to the number of injected files in the Hierarchical Search attack, which is $\lceil \frac{|R|}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$, and the number of injected files in the Decoding attack, which is $|R|$, we can see that our approach requires fewer injected files. This indicates that our attack achieves efficient query recovery with a reduced number of injected files compared to the Hierarchical Search and Decoding attacks.

### Query Recovery

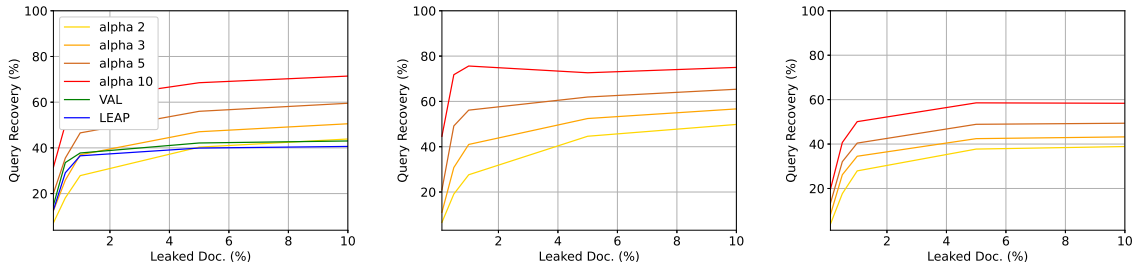
If we visualize the data from Table 7.1 in the form of a graph, we obtain Figure 7.3, which represents the query recovery results. This graph provides a clearer visual understanding of the performance of the proposed attack under different settings of data leakage and $\alpha$ value for the three datasets.

Consistently, higher values of $\alpha$ lead to constantly higher query recovery rates. This is because the recovered queries include not only those inferred from matched documents but also those directly recovered through keyword injection. As the $\alpha$ value increases, the size of the injected keyword set $W_i'$ becomes larger, resulting in improved query recovery rates.

Recalling the results of document recovery from our last experiment in Figure 4.10 and 4.11, all strategies encountered a low recovery rate. Among all the strategies, the highest recovery rate in all three datasets was achieved by strategy V2 (choosing global low-frequent keywords). However, even with this strategy, the recovery rate struggled to surpass 55% in the Enron and Lucene datasets. In the case of the Wikipedia dataset, it only slightly exceeded 45% at the leakage level of 10%.

Upon observing Figure 7.3, it is evident that the query recovery rates have significantly improved. The highest query recovery rate in the Enron dataset has increased to lightly surpass 70%, while in

Lucene, this number is closer to 80%. Even in the Wikipedia dataset, with the largest number of keywords and the highest proportion of inert keywords, the highest query recovery rate approaches 60%.




**(a)** Enron **(b)** Lucene **(c)** Wiki

**Figure 7.3:** Query Recovery Performance Over the Entire Keyword Universe

**Comparing with LEAP and VAL**

In Figure 7.4a, a direct comparison of query recovery is presented between our attack and the VAL and LEAP attacks. The keyword universe used for this comparison is extracted from the entire dataset of Enron.

Overall, our attack outperforms VAL and LEAP in terms of query recovery. While the recovery rates of LEAP (blue line) and VAL (green line) are strictly limited by the proportion of active keywords in Enron, which is 43.5%, our attack can already surpass this limitation and reach a higher recovery rate.

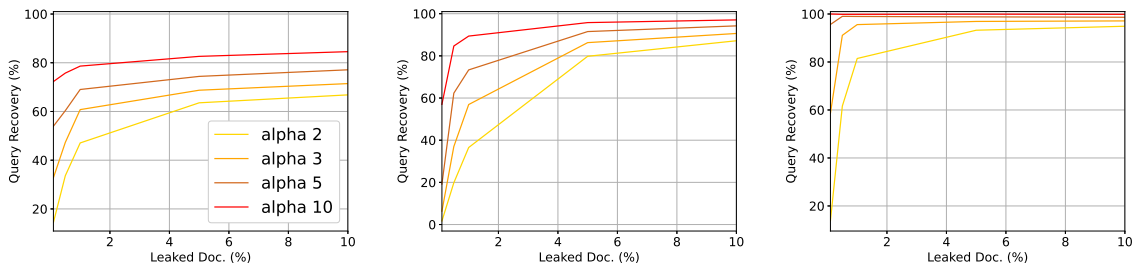
With $\alpha$ values of 5 and 10 for all leakage levels, our attack outperforms VAL and LEAP. Even with $\alpha$ set to 3, our attack surpasses both attacks after a leakage of approximately 1%. Additionally, even $\alpha$ 2 surpasses LEAP starting from a 5% leakage level and exhibits a trend of surpassing VAL before reaching a 10% leakage level.

### Active Keyword Recovery

In Figure 7.4, we provide a graph that illustrates the performance of our attack in inferring the underlying active keywords of queries. The graph shows the proportion of correctly recovered active keywords under different settings of $\alpha$ values and data leakage levels within the three datasets.

The current method we employ to infer active keywords involves a one-round match of unique occurrence patterns between keywords and queries. It is remarkable to observe that, when using this one-round match with an $\alpha$ value of 10, the active keywords in the Wikipedia dataset can be consistently and perfectly recovered, regardless of the data leakage level.

Overall, we observe that all $\alpha$ values can achieve a query recovery rate higher than 90% in the Wikipedia dataset, starting from a data leakage of 5%. In the Lucene dataset, even with an $\alpha$ value of 2, the query recovery rate is able to exceed 80% from a leakage level of 5%. However, the active keyword recovery of the attack performs worst in the Enron dataset, where even with an $\alpha$ value of 10, the recovery rate of active keywords is only closer to 85% at a leakage level of 10%.




**(a)** Enron **(b)** Lucene **(c)** Wiki

**Figure 7.4:** Query Recovery Performance Over Active Keywords

## 7.3. Discussion

The updated attack has achieved satisfactory document recovery rates, comparable to the LEAP attack and slightly lower than the VAL attack in the Enron dataset.

By examining the rate of return of query recovery, we confirmed that our attack retains the ability of the amplifier effect, which helps file injection attacks to significantly reduce the size of injected documents.

Furthermore, our attack has proven to be effective in assisting state-of-the-art inference attacks (LEAP and VAL) in overcoming the limitations set by the properties of inert keywords.

However, our recovery rate of active keywords remains at a relatively low level in the Enron dataset. This could be due to our one-round keyword match method, which may not be sufficient to fully capture the underlying patterns of active keywords.

Furthermore, to validate the robustness of our results, we conducted the experiments again with 20 repeated runs for each experiment. Additionally, we included the standard deviation of the recovery rates in the results to assess their stability. The verification results are presented in Appendix B. As expected, all results obtained from this verification confirmed our current analysis.

# 8

# Conclusion

## 8.1. Conclusion

In conclusion, our research has presented a novel approach to SSE attacks by integrating state-of-the-art inference and file injection techniques. The motivation for combining these two techniques stems from the limitations faced by both inference and file injection attacks:

From the perspective of file injection attacks, our assumption that the attacker has knowledge of partial data leakage is grounded in the reality that data leakage is often unavoidable in real-life scenarios. By simulating this practical situation, we aim to assess the potential improvements that can be achieved when the attacker possesses such resources.

As for inference attacks, the integration with file injection attacks addresses the inherent limitation they face in being unaware of a significant number of inert keywords that cannot be inferred using access patterns.

The attack comprises three stages: keyword selection, file injection, and recovery. Within this new attack framework, we introduced four distinct strategies for the keyword selection stage.

The first three strategies were designed when we did not realize the limitation faced by inference attacks, and we would like to build a file injection attack with an amplifier effect. The basic idea is to choose multiple keywords from the known keyword set, each carrying more information to enable document recovery, and then recover additional keywords using the matched documents. We classified keywords based on their frequency, as we intuited that low-frequency keywords contain more valuable information. Our target at this stage was to achieve a comparable query recovery result as proposed in LEAP and VAL, aiming to surpass 90%.

While our results preliminarily proved the possibility of achieving an amplifier effect when partial data leakage is allowed for file injection attacks, the obtained query recovery rate was lower than expected, with the best rate surpassing 60% in the Lucene dataset under a data leakage of 10%.

The last strategy was proposed after introducing a new classification of keywords, grouping them based on their occurrence patterns as active or inert keywords. For document recovery, we selected multiple inert keywords from each document. The attack objective became using injected inert keywords to recover as many active keywords as possible, thereby overcoming the current limitation of inference attacks that can only infer active keywords. We verified this limitation by reproducing the VAL attack with a larger keyword universe and confirmed that its query recovery result could no longer surpass the proportion of active keywords in the keyword universe.

The result of this last proposed strategy yielded the expected outcome, with the number of recovered queries surpassing the number of active keywords. The best recovery rate for active keywords was observed within the Wikipedia dataset, achieving 100% with an $\alpha$ value of 10, irrespective of the data leakage level. Although this remarkable recovery rate for active keywords did not occur with other datasets, it can be understood as a trade-off, as our attack only performs one round of matching in both

document and query recovery stages, sacrificing higher recovery rates for shorter computation time. Furthermore, the attack met the objective of recovering more queries with a single injected keyword, with an average of 1.3 keywords recovered even in the worst-case scenario.

As a result, we consider our proposed attack to be a valid improvement for existing SSE attacks and provide a satisfactory answer to our initial research question.

One important observation is that datasets with a higher proportion of inert keywords are more suitable targets for our attack. This finding suggests that future SSE attacks should consider dataset type or distribution to identify the most applicable targets.

Additionally, from a cybersecurity research perspective, it is reassuring to see that commonly used access-hiding countermeasures remain robust against this newly proposed attack.

Moreover, the proposed classification of keywords offers valuable insights for further advancements in SSE-related research.

Looking ahead, future research in SSE attacks could focus on exploring novel techniques to recover inert keywords, considering the outstanding performance of inference attacks in recovering active keywords. Furthermore, SSE schemes could explore ways to enhance resistance against attacks that aim to recover active keywords, thereby optimizing computational and storage resources that might be spent on inert keywords.

## 8.2. Future work

After conducting a self-evaluation, we recognize that there are several points of improvement that we haven't been able to explore within our current project due to time limitations. These areas could be conducted as future work:

**Comparison with LEAP and VAL using Lucene and Wikipedia:** We attempted to run LEAP and VAL with the keyword universe in Wikipedia during our experiment, but the significant computation time hindered us from completing it. It would be beneficial to conduct further research and obtain a more comprehensive evaluation by including Lucene and Wikipedia datasets in the comparison.

**Insufficient evaluation under real dynamic SSE schemes:** In our current experiment, we assumed a relatively ideal scenario where the users do not alter the dataset during the file injection phase. However, in real-world scenarios, file additions and deletions might impact the performance of the proposed attack. Further evaluation under dynamic SSE schemes is necessary to validate the attack's effectiveness in real-world settings.

**Incorporating volume pattern and recursive matching algorithm:** As demonstrated by VAL and LEAP, a higher query recovery rate can be achieved by building a loop between document and query recovery and incorporating additional leakage patterns, such as volume pattern. Further research can explore the integration of volume pattern and recursive matching algorithms to enhance the attack's performance.

Additionally, there are other areas of improvement that we have proposed in Chapter 3 but have not had the opportunity to explore in this research. For instance, investigating the possibility of query recovery under the assumption that only partial content in each leaked document is known by the attacker, and continuously exploring SSE attacks with higher resistance to existing countermeasures would be valuable directions for future research.

# References


[1] Rakesh Agrawal et al. “Order Preserving Encryption for Numeric Data”. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’04. Paris, France: Association for Computing Machinery, 2004, pp. 563–574. isbn: 1581138598. doi: 10.1145/1007568.1007632.

[2] Mihir Bellare, Alexandra Boldyreva, and Adam O’Neill. “Deterministic and Efficiently Searchable Encryption”. In: *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO’07. Santa Barbara, CA, USA: Springer-Verlag, 2007, pp. 535–552. isbn: 3540741429.

[3] Laura Blackstone, Seny Kamara, and Tarik Moataz. “Revisiting leakage abuse attacks”. In: *Cryptology ePrint Archive* (2019). url: https://eprint.iacr.org/2019/1175.

[4] Dan Boneh et al. “Public key encryption with keyword search”. In: *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2004, pp. 506–522.

[5] Raphael Bost. *Sophos - Forward Secure Searchable Encryption*. Cryptology ePrint Archive, Paper 2016/728. 2016. doi: 10.1145/2976749.2978303.

[6] Raphael Bost and Pierre-Alain Fouque. “Thwarting Leakage Abuse Attacks against Searchable Encryption - A Formal Approach and Applications to Database Padding”. In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 1060.

[7] David Cash et al. *Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation*. Cryptology ePrint Archive, Paper 2014/853. 2014. doi: 10.14722/ndss.2014.23264.

[8] David Cash et al. “Leakage-Abuse Attacks Against Searchable Encryption”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS ’15. Denver, Colorado, USA: Association for Computing Machinery, 2015, pp. 668–679. isbn: 9781450338325. doi: 10.1145/2810103.2813700.

[9] Melissa Chase and Seny Kamara. “Structured Encryption and Controlled Disclosure”. In: *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*. Vol. 6477. Lecture Notes in Computer Science. Springer, 2010, pp. 577–594. doi: 10.1007/978-3-642-17373-8_33.

[10] 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.

[11] Reza Curtmola et al. “Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS ’06. Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 79–88. isbn: 1595935185. doi: 10.1145/1180405.1180417.

[12] Amit Sahai Dan Boneh and Brent Waters. “Functional Encryption: Definitions and Challenges”. In: *Theory of Cryptography*. TCC ’11. Berlin, Heidelberg: Springer, 2011, pp. 253–273. doi: 10.1007/978-3-642-19571-6_16.

[13] *Data Leakage: Common Causes, Examples  Tips for Prevention*. url: https://www.bluevoyant.com/knowledge-center/data-leakage-common-causes-examples-tips-for-prevention#:~:text=What%20Is%20Data%20Leakage%3F,chat%20rooms%2C%20and%20other%20communications..

[14] The Apache Software Foundation. *Apache: Mail archieves of lucene*. 1999. url: https://lists.apache.org/#lucene.

[15] Craig Gentry. “Fully Homomorphic Encryption Using Ideal Lattices”. In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC ’09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 169–178. isbn: 9781605585062. doi: 10.1145/1536414.1536440.

[16] Oded Goldreich and Rafail Ostrovsky. “Software Protection and Simulation on Oblivious RAMs”. In: *J. ACM* 43.3 (May 1996), pp. 431–473. issn: 0004-5411. doi: 10.1145/233551.233553.

[17] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. “Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation”. In: *Network and Distributed System Security Symposium*. 2012.

[18] 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.

[19] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. “Dynamic Searchable Symmetric Encryption”. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS ’12. Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, pp. 965–976. isbn: 9781450316514. doi: 10.1145/2382196.2382298.

[20] Dalia Khader. *Public Key Encryption with Keyword Search based on K-Resilient IBE*. Cryptology ePrint Archive, Paper 2006/358. https://eprint.iacr.org/2006/358. 2006. url: https://eprint.iacr.org/2006/358.

[21] Kee Sung Kim et al. “Forward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1449–1463. isbn: 9781450349468. doi: 10.1145/3133956.3133970.

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680. issn: 00368075. doi: 10.1126/science.220.4598.671.

[23] Y.A.M. Kortekaas. *Access Pattern Hiding Aggregation over Encrypted Databases*. Oct. 2020. url: http://essay.utwente.nl/83874/.

[24] Steven Lambregts. “Revisit Attacks on Searchable Symmetric Encryption: Explore More, Reveal More”. In: (2022). url: http://resolver.tudelft.nl/uuid:ec72afb6-ad96-4357-94da-ad2c7bb2a6fd.

[25] Hao Liu et al. “Vaccine:: Obfuscating Access Pattern Against File-Injection Attacks”. In: *2019 IEEE Conference on Communications and Network Security (CNS)*. 2019, pp. 1–9. doi: 10.1109/CNS.2019.8802803.

[26] William W. Cohen MLD. *Enron Email Datasets*. 2015. url: https://www.cs.cmu.edu/~enron/.

[27] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. *Dynamic Searchable Encryption via Blind Storage*. Cryptology ePrint Archive, Paper 2014/219. 2014. url: https://eprint.iacr.org/2014/219.

[28] Jianting Ning et al. “LEAP: Leakage-Abuse Attack on Efficiently Deployable, Efficiently Searchable Encryption with Partially Known Dataset”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 2307–2320. isbn: 9781450384544. doi: 10.1145/3460120.3484540.

[29] Jianting Ning et al. “Passive Attacks Against Searchable Encryption”. In: *IEEE Transactions on Information Forensics and Security* 14.3 (2019), pp. 789–802. doi: 10.1109/TIFS.2018.2866321.

[30] Martin F Porter. “An algorithm for suffix stripping”. In: *Program* 14.3 (1980), pp. 130–137.

[31] David Pouliot and Charles V. Wright. “The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption”. In: CCS ’16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 1341–1352. isbn: 9781450341394. doi: 10.1145/2976749.2978401.

[32] Y. Zhang R. Du and M. Li. “Database Padding for Dynamic Symmetric Searchable Encryption”. In: *Security and Communication Networks*. Vol. 2021. Dec. 2021, p. 9703969. doi: 10.1155/2021/9703969.

[33] Bruce Schneier. *Secrets Lies: Digital Security in a Networked World*. 1st. USA: John Wiley Sons, Inc., 2000. isbn: 0471253111.

[34] Zhiwei Shang et al. “Obfuscated Access and Search Patterns in Searchable Encryption”. In: *Proceedings 2021 Network and Distributed System Security Symposium* (2021). doi: 10.14722/ndss.2021.23041.

[35] David Shapiro. *Daveshap/plaintextwikipedia: Convert Wikipedia database dumps into plaintext files*. 2021. url: https://github.com/daveshap/PlainTextWikipedia.

[36] Dawn Xiaoding Song, D. Wagner, and A. Perrig. “Practical techniques for searches on encrypted data”. In: *S&P 2000*. IEEE, 2002. doi: 10.1109/SECPRI.2000.848445.

[37] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. “Practical dynamic searchable encryption with small leakage”. In: *Cryptology ePrint Archive* (2013).

[38] Edward Loper Steven Bird Ewan Klein. *Natural language processing with python*. 2009.

[39] S. Umeyama. “An eigendecomposition approach to weighted graph matching problems”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.5 (1988), pp. 695–703. doi: 10.1109/34.6778.

[40] David C. Uthus and David W. Aha. “The Ubuntu Chat Corpus for Multiparticipant Chat Analysis”. In: *AAAI Spring Symposium: Analyzing Microtext*. 2013.

[41] Andrew Wintenberg et al. “A Dynamic Obfuscation Framework for Security and Utility”. In: *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. 2022, pp. 236–246. doi: 10.1109/ICCPS54341.2022.00028.

[42] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. “A Path Following Algorithm for the Graph Matching Problem”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.12 (2009), pp. 2227–2242. doi: 10.1109/TPAMI.2008.245.

[43] Xianglong Zhang et al. *High Recovery with Fewer Injections: Practical Binary Volumetric Injection Attacks against Dynamic Searchable Encryption*. 2023. arXiv: 2302.05628 [cs.CR].

[44] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. “All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption”. In: *Proceedings of the 25th USENIX Conference on Security Symposium*. SEC’16. Austin, TX, USA: USENIX Association, 2016, pp. 707–720. isbn: 9781931971324.

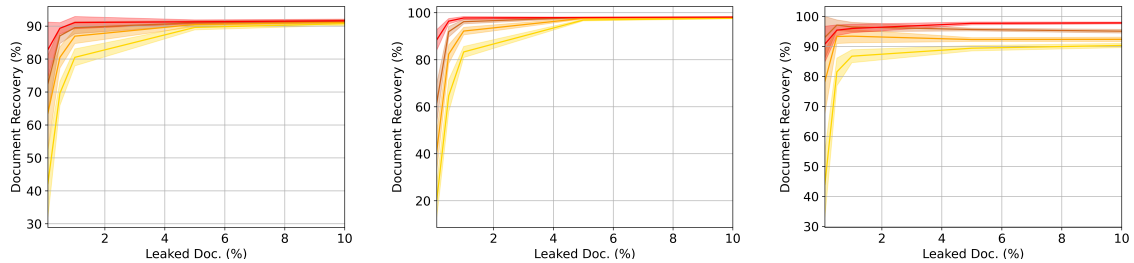
# A

# Code: Downloading Lucene Mailing List Archives

```python
import urllib.request
import os
from tqdm import tqdm

# Specify the URL template for the Lucene mailing list archives
url_template = "http://mail-archives.apache.org/mod_mbox/lucene-java-user/{year}{month}.mbox"

# Specify the directory where the mbox files will be stored
directory = ""

# Create the directory if it doesn't exist
if not os.path.exists(directory):
    os.makedirs(directory)

# Download mbox files for the years 2001 to 2011
for year in tqdm(range(2001, 2012)):
    if year == 2001:
        months = [9, 10, 11, 12]
    else:
        months = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
    for month in months:
        url = url_template.format(year=year, month=f'{month:02d}')
        filename = f'{year}-{month:02d}.mbox'
        filepath = os.path.join(directory, filename)
        urllib.request.urlretrieve(url, filepath)
```

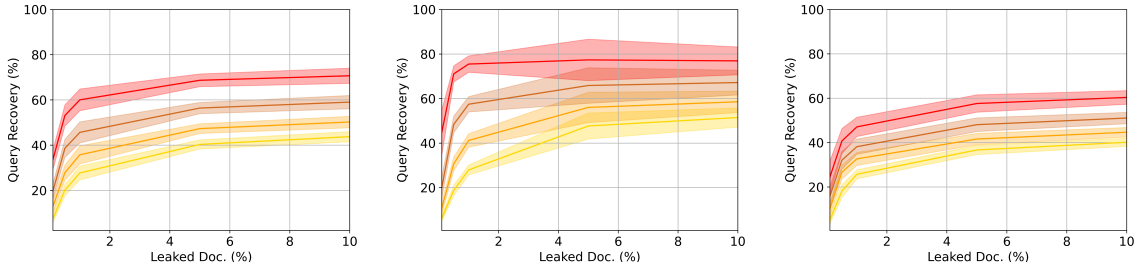
# B

# Result Verification



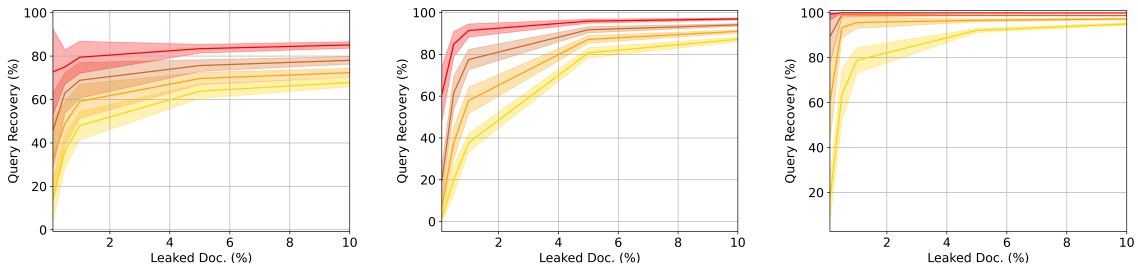

**(a)** Enron **(b)** Lucene **(c)** Wikipedia

**Figure B.1:** Document Recovery performance




**(a)** Enron **(b)** Lucene **(c)** Wiki

**Figure B.2:** Query Recovery Performance Over the Entire Keyword Universe




**(a)** Enron **(b)** Lucene **(c)** Wiki

**Figure B.3:** Query Recovery Performance Over Active Keywords