Inject Less, Recover More

Unlocking the Potential of Document Recovery in Injection Attacks Against SSE

Zhang, Manning; Shi, Zeshun; Chen, Huanhuan; Liang, Kaitai

DOI
10.1109/CSF61375.2024.00029

Publication date
2024

Document Version
Final published version

Published in
Proceedings of the 2024 IEEE 37th Computer Security Foundations Symposium (CSF)

Inject Less, Recover More: Unlocking the Potential of Document Recovery in Injection Attacks Against SSE

Manning Zhang*†, Zeshun Shi*†, Huanhuan Chen*†, and Kaitai Liang*
*Cyber Security Group, Delft University of Technology, 2628 CD, the Netherlands
Email: zhangmanning2015@gmail.com, {z.shi-2, h.chen-2, kaitai.liang}@tudelft.nl

2024 IEEE 37th Computer Security Foundations Symposium (CSF) | 979-8-3503-6203-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/CSF61375.2024.00029

***Abstract*—Searchable symmetric encryption has been vulnerable to inference attacks that rely on uniqueness in leakage patterns. However, many keywords in datasets lack distinctive leakage patterns, limiting the effectiveness of such attacks. The file injection attacks, initially proposed by Cash et al. (CCS 2015), have shown impressive performance with 100% accuracy and no prior knowledge requirement. Nevertheless, this attack fails to recover queries with underlying keywords not present in the injected files. To address these limitations, our research introduces a novel attack strategy called LEAP-Hierarchical Fusion Attack (LHFA) that combines the strengths of both file injection attacks and inference attacks. Before initiating keyword injection, we introduce a new approach for inert/active keyword selection. In the phase of selecting injected keywords, we focus on keywords without unique leakage patterns and recover them, leveraging their presence for document recovery. Our goal is to achieve an amplified effect in query recovery. We demonstrate a minimum query recovery rate of 1.3 queries per injected keyword with a 10% data leakage of a real-life dataset, and initiate further research to overcome challenges associated with non-distinctive keywords.**

***Index Terms*—Searchable symmetric encryption, Inference attack, File injection attack, Access pattern**

I. Introduction

Searchable Symmetric Encryption (SSE), introduced by Song et al. in 2000 [1], addresses the need for efficient search operations on encrypted datasets while maintaining data confidentiality. With the increasing adoption of cloud technology, there is a growing demand for searchable functionality without compromising data security [2]. SSE enables users to encrypt their data locally and upload it to a server that may be curious but honest. Later, users can retrieve specific documents by providing encrypted search tokens, ensuring that no plaintext information is leaked in the process. Since its inception, SSE techniques have undergone significant enhancements and refinements through extensive research efforts.

SSE Attacks: Despite the security guarantees provided by encryption, there are potential leakages within SSE schemes, as discussed in Section II. To exploit these vulnerabilities, SSE attacks have been developed and can be broadly categorized into two main types: passive (inference) attacks and active (file injection) attacks.

Inference attacks [3], [4], [5], [6], [7], [8], [9] have further diversified into two subtypes: leakage abuse attacks and similar data attacks. These attacks primarily exploit leaked plaintext, either from the same dataset or a similar dataset, as well as the relationship between search tokens and server responses to extract hidden knowledge. The IKK attack [3] serves as a notable example, marking the initial exploration of SSE attacks. Subsequent research on inference attacks has showcased advancements in query recovery accuracy while minimizing the amount of data leakage required.

File injection attacks [4], [6], [10] constitute another branch of SSE attacks. These attacks involve tricking users into uploading files containing specific keywords designed by the attacker. The attacker learns the underlying encryption files of those injected documents by observing their arriving time or volume, and with this knowledge, the response file returned for a query allows immediate determination of the underlying keyword. File injection attacks are characterized by high accuracy and do not require prior knowledge of leaked documents. Recent investigations have aimed to minimize the number of injected files needed for successful injection.

Motivation: During our revisitation of SSE attacks, we have identified certain limitations and raised questions.

The experiments of existing inference attacks often employ a relatively limited chosen keyword universe in their experiments, such as the 5,000 most common keywords [6], [8], [9]. This limitation is due to computational constraints and the need for fair comparisons with previous attacks that used the same keyword universe. However, the accuracy of these attacks when applied to larger-scale keyword universes has not been thoroughly evaluated. Our study conducted tests using the VAL attack with a wider keyword universe, revealing a significant decrease in accuracy from 99% to 42%, as shown in Section III. This decrease can be attributed to the presence of keywords inherently lacking a unique appearance pattern, making their recovery challenging through inference.

In the domain of file injection attacks, the conventional approach to minimizing the number of injected files focuses on exploring new arrangements of keywords within the

[†] The first three authors contributed equally to this work.

DOI 10.1109/CSF61375.2024.00029

injected files. However, drawing from our insights gained through the study of inference attacks, we raise an important question: Is it possible to infer queries based on the recovered injected keywords? In other words, we aim to design an attack that acts as an amplifier, recovering not only the injected keywords but also additional ones.

Our Contribution: Our study makes significant contributions in two main areas. Firstly, we conduct a thorough analysis of the dataset and introduce a novel classification of keywords based on their possession of a unique leakage pattern. This classification sheds light on the inherent limitations of inference attacks, providing a comprehensive understanding of their effectiveness and challenges. Secondly, drawing insights from LEAP and Hierarchical Search attacks, we merge their strengths to propose a novel attack method known as the LEAP-Hierarchical Fusion Attack (LHFA). This innovative attack introduces an amplification effect in file injection attacks while overcoming the inherent limitations of inference attacks.

We evaluate the performance of our proposed attack using three real-life datasets and compare the results with LEAP [8] and VAL [9] to demonstrate that our approach surpasses the query accuracy limitations of inference attacks. Furthermore, we compare the ratio between injected and recovered keywords with Hierarchical Search [11] and Decoding [6] attacks to validate the successful achievement of the amplification effect.

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.

II. Background Knowledge

Within the realm of Searchable Symmetric Encryption (SSE), data takes the form of a collection of documents denoted as D, where each document is composed of a set of keywords represented as W. SSE facilitates the searchability of encrypted data through a series of steps, encompassing encryption, query generation, and search procedures.

The encryption process initiates with the client locally encrypting the dataset D into an equivalent set of encrypted documents using a private key. Each encrypted document contains a set of encrypted keywords referred to as queries, denoted as Q. Subsequently, the encrypted dataset, denoted as ED, is transmitted to the server, which is assumed to be honest but inquisitive.

The phase of query generation precedes the actual search operation. When the client desires to locate a specific document, a query is generated locally utilizing the private key. This query is then dispatched to the server. In response, the server provides the encrypted documents that contain the query. The client then decrypts these documents to extract the desired content. In dynamic SSE [12], [13], additional functionalities for adding and deleting data are incorporated. An update algorithm is used to inform the server about changes when data is added or removed, ensuring effective maintenance and dynamic updates of the outsourced dataset.

Both basic and dynamic SSE schemes do not expose any plaintext information to parties other than the client, thereby maintaining a level of security.

A. Leakage in SSE

Despite the relative security of encryption in SSE, two types of leakage can be exploited by SSE attacks.

Leakage Pattern: Leakage patterns can be discerned from the communication between the client and the server without exposing the actual content of the communication. These patterns encompass the informational links between queries and encrypted documents, thereby characterizing the identity of a query, potentially allowing it to be distinguished from other queries. To illustrate, access patterns and volume patterns stand out as two common instances of leakage patterns frequently employed in various SSE attack scenarios. Access patterns unveil the relationships between queries and server responses by defining a query through a set of identifiers for encrypted documents that contain it. Volume patterns scrutinize the volume of encrypted documents to identify queries. By scrutinizing these patterns, attackers can gain insights into establishing ownership connections between encrypted documents and queries and consequently extract more sensitive information through various attack mechanisms, as elaborated in the following section.

Data Leakage: Data leakage represents an inescapable and formidable challenge, not limited to SSE systems alone but also prevalent in most encryption systems. Achieving absolute control and prevention of data leakage proves to be a difficult task since human factors are involved. It forms the basis for inference attacks. The extent of data leakage is influenced by encryption strength as well as user awareness and behavior. Even if SSE scheme promises absolute security in encryption, vulnerabilities such as attacker access to local devices or user carelessness can lead to a data leakage.

B. Related Work

Table I provides a succinct overview of several existing SSE attacks that we have studied.

Our investigation into inference attacks has unveiled a crucial factor significantly impacting the final query recovery rate: the attacker's precision in keyword identification.

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.

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. This is because inference attacks are still working towards reaching the same level of success in terms of high recovery accuracy with minimal data leakage.

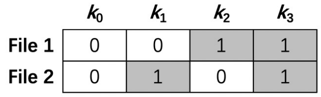
The introduction of the Binary Search attack marked a significant breakthrough in reducing the required number of injected documents for the attack. Subsequent attacks like Decoding and Binary Variable-Parameter Attack (BVA) have explored volume patterns, further enhancing the practicality and efficiency of file injection attacks. Nevertheless, the study of BVA revealed a trade-off between the number of injected documents and the final query recovery rate.

In our comprehensive review of each attack, it became evident that each one marks a noteworthy milestone, introducing fresh ideas and approaches to the realm of SSE attacks. Researchers continually endeavor to strike the optimal equilibrium between the attack's assumption and the query recovery rate while tailoring these attacks to a wide range of leakage scenarios.

We will provide a more comprehensive description of four attacks that have significantly influenced our study.

Binary Search Attack [11]: This attack leverages the binary search concept. Each injected file contains exactly half of the keywords from the injected keyword universe K, resulting in a maximum injected file size of $\lceil \log |K| \rceil$. The file i consists of keywords from K whose ith bit is 1, ensuring that each keyword has a unique access pattern among the injected files. Through analysis of the presence or absence of keywords in the returned files, the attacker can deduce the queried keyword with a 100% accuracy rate.

As an example, consider a keyword universe with four keywords represented as k_0, k_1, k_2, and k_3. These keywords are assigned to two files $File$ 1 and $File$ 2, as illustrated in Figure 1, where 1 denotes the presence of the corresponding keyword and 0 indicates its absence. If, for instance, only encrypted $File$ 1 is present in the returned response, the attacker can immediately deduce that the underlying keyword for the corresponding query is k_2, as it is the only keyword exclusively contained in $File$ 1.



	k_0	k_1	k_2	k_3
File 1	0	0	1	1
File 2	0	1	0	1

Fig. 1. Example of Binary Search Attack

Hierarchical Search Attack [11]: The Hierarchical Search attack addresses the drawback of the Binary Search attack, where the conspicuous size of injected files can be detected. This method involves partitioning the keyword universe into subsets. Subsequently, the Binary Search attack is applied to pairs of subsets individually. This approach balances the injected file size and accuracy in query recovery and results a set of injected files with maximum size of $\lceil \frac{|K|}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$, where T indicates the length of limitation of each injected keyword subset.

LEAP Attack [8]: LEAP introduces a comprehensive approach by considering the uniqueness of both keywords and documents in the dataset. It compares the unique number of keywords in each document and matches encrypted documents with leaked ones. This process establishes a reference standard for allocating uniqueness to keywords found in the matched documents. By leveraging the identified keywords, LEAP enables the discovery of additional document matches, creating a positive feedback loop. This loop continues until all target keywords and leaked documents with uniqueness in access pattern are successfully matched.

VAL Attack [9]: VAL builds upon the foundation established by LEAP and incorporates the use of volume patterns. This approach enhances the distinctiveness of documents and keywords, resulting in higher accuracy in both document and query recovery, and resistance against countermeasures targeting single leakage patterns.

III. Revisit and Discovery

This section aims to provide the rationale and necessity for combining inference attacks and file injection attacks in our proposed attack. We discuss the limitations of file injection attacks and the potential for leveraging leaked data. Furthermore, we probe into the inherent limitation contributing to unexpected outcomes observed in the replication of the VAL attack, and this limitation is also shared by all existing inference attacks. We also contemplate potential remedies to surmount this limitation.

A. File Injection Attack

File injection attacks [6], [10], [11] have demonstrated two notable strengths: 100% query recovery accuracy on injected queries and the absence of a requirement for prior knowledge. These attacks strategically organize target keywords within injected files, establishing them as the most powerful type of attack in the field of SSE. However, these strengths impose an invisible limitation on file injection attacks.

The emphasis on achieving 100% accuracy restricts further exploration in improving the accuracy of file injection attacks. Specifically, the maximum size of the recoverable query is constrained by the size of the injected keyword universe. The strong assumption of not relying on data leakage overlooks the potential utilization of leaked information that unavoidably occurs in real-life scenarios.

This leads us to ponder whether it is viable to formulate an attack strategy that unveils the outcomes of attackers employing file injection techniques to enhance their retrieval of

TABLE I
OVERVIEW OF SSE ATTACKS

IN Attack[1]	**Leakage Pattern**[2]	**Data Leakage**	**Approach**	**Target**
IKK [3]	AP	All	Keyword Co-occurrence	Query
Count [4]	AP,RLP	Partial	Keyword Co-occurrence, Response length	Query
Shadow Nemesis [5]	AP	Similar	Keyword Co-occurrence	Query
Subgraph [6]	ALP	Partial	Subset	Query
LEAP [8]	AP	Partial	Document Co-occurrence, Length	Document, Query
VAL [9]	AP,VP	Partial	Document Co-occurrence, Length, Volume	Document, Query
FI Attack[1]	**Leakage Pattern**	**Injection Size**[3]		**Target**
Binary Search [11]	AP	$\lceil \log \lvert K \rvert \rceil$		Query
Hierarchical Search [11]	AP	$\lceil \frac{\lvert K \rvert}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$		Query
Decoding [6]	VP	$\lvert K \rvert$		Query
BVA [10]	VP	$\lceil \log \lvert K \rvert \rceil$		Query
Our Attack	**Leakage Pattern**	**Data Leakage**	**Injection Size**[4]	**Target**
IN and FI	AP	Partial	$\lceil \frac{\frac{\lvert K \rvert}{\beta}}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$	Document, 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] 'AP', 'VP', 'RLP', and 'ALP' correspond to Access Pattern, Volume Pattern, Response Length Pattern, and Atomic Leakage Pattern, respectively. Specifically, RLP refers to the number of returned files, while ALP encompasses any pattern that exposes a characteristic of each matching document, including both AP and VP.
[3] $|K|$ denotes the number of injected keywords and T means the length limitation of each document.
[4] β represents the number of queries that can be recovered per injected keyword, and its value varies depending on the specific configuration or setting.

encrypted data while allowing assumptions based on partially leaked information in plaintext.

B. Inference Attack

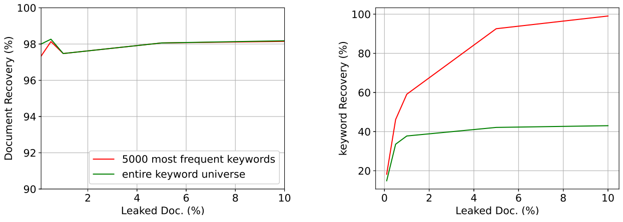
Prior to showcasing the limitations encountered by inference attacks, we aim to present the results of our replication of the VAL attack.

Reproduction. The aim of this replication is to validate the results of the proposed inference attacks across a more diverse range of keyword universes. The selection of the VAL attack was motivated by its superior performance compared to its foundational predecessor, LEAP, which had previously held the most notable query recovery outcome among various inference attacks prior to the introduction of VAL. The reproduction was executed using the Enron dataset [14] as the testing dataset.

In Figure 2, the red lines illustrate the outcomes obtained when employing the top 5000 most frequent keywords as the keyword universe, mirroring the settings used in the VAL attack. The results reaffirmed similar conclusions as those proposed by VAL: document recovery consistently achieved an accuracy rate of 98%, and query recovery accuracy consistently exceeded 90%, even in scenarios featuring a 5% data leakage. Furthermore, as the data leakage increased to 10%, the accuracy of query recovery approached 100%.

However, when we expanded the keyword universe from the most common 5000 keywords to include all known keywords in the leaked documents, a significant increase in running time was first observed. This heightened runtime can be attributed to the recursive implementation of document recovery and query recovery steps within the VAL attack logic, with its duration notably impacted by the scale of the input query set.

Additionally, notable deviations in the results of query recovery were identified, as depicted by the green line in Figure 2b. 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%.



(a) Document Recovery (b) Document Recovery

Fig. 2. Results of VAL reproduction

To unravel the enigma surrounding this deviation attributed to the enlargement of the keyword universe, we embarked on an extensive investigation. This encompassed a meticulous examination of the code used, a re-execution of the experiment, and also a detailed scrutiny of the inherent characteristics of the keywords themselves. This exploration prompted us to put forward a new classification of keywords, a topic that had not been elaborated upon in previous research.

Keyword classification. While previous research has primarily focused on keyword frequency, alternative classifications of keywords have received limited attention. To address this gap, we propose a classification into two categories: inert keywords and active keywords, named after the chemical terms "inert gas" and "active gas," respectively. Active keywords exhibit a distinctive appearance pattern that enables their effective differentiation from all other keywords. In contrast, inert keywords lack a discernible appearance pattern; they often share the same set of documents with one or more other keywords.

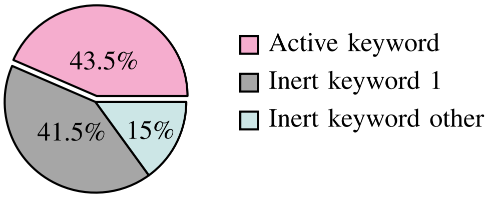


Fig. 3. Keyword distribution structure of Enron. The pink area 'Active keyword' in the pie charts represents the proportion of active keywords in each dataset, while 'Inert keyword 1' represents inert keywords that appeared in only one document. 'Inert keyword other' indicates inert keywords that appeared in more than one document.

Upon analyzing the dataset using this new classification, we discovered that the proportion of inert keywords is greater than initially anticipated. Figure 3 illustrates this distribution. 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 2b, which approximates the 43% mark.

The selection of the 5000 most common keywords used in VAL exhibits a high proportion of active keywords. Using the same classification approach, we found that 4996 out of the 5000 keywords were identified as active keywords, resulting in an almost 100% classification rate. Consequently, the experimental results reported in VAL achieved high accuracy due to the dominance of active keywords within this subset.

The properties of inert keywords, in conjunction with the comparison results, highlight the significant challenge of achieving a relatively high query recovery accuracy for inference attacks operating within the context of the entire keyword universe, even without any countermeasures in place.

To surmount this challenge, inference attacks must either unearth alternative leakage patterns capable of discerning between two queries that appear identical in documents or enlist the support of other SSE attacks that enable the creation of artificial distinctive patterns for keywords. For instance, file injection attacks have the capability to generate unique patterns for specific target keywords.

IV. The Proposed Attack: LHFA

In this section, we explain how our new attack works. The attack is designed with the objective of assisting inference attacks in addressing the limitation posed by inert keywords. Additionally, it serves as an experimental solution to assess the feasibility of generating an amplification effect for file injection attacks. This attack is named the LEAP-Hierarchical Fusion Attack (LHFA) as it is constructed upon the foundation of the Hierarchical Search attack [11] while integrating the concept put forth by the LEAP attack [8].

A. Notation

In our proposed attack, we consider a scenario with n server documents (ed_i) and n' leaked files (d_i). The query universe extracted from the server documents consists of m queries (q_i), while m' keywords (k_i) can be extracted from the leaked files. It is important to note that the index i does not imply a direct correspondence between d_i and ed_i, or between k_i and q_i. Table II provides a list of the notations used in our attack, and their explanations will be provided subsequently.

TABLE II
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, $W' = \{k_1, ..., k_{m'}\}$ where $W' \subseteq W$
W'_{ac}	Active keyword set in known keywords, $W'_{ac} = \{k_1, ..., k_{m'_{ac}}\}$ where $W'_{ac} \subseteq W'$
W'_i	Chosen keyword set, $W' = \{k_1, ..., k_{m'_i}\}$ where $W'_i \subseteq W'$
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
α	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

B. Intuition

The research conducted in LEAP has shed light on a valuable insight: the strategic inclusion of a document recovery step as an intermediate objective holds the potential to enhance query recovery substantially.

Our intuition is applying this insight to file injection attacks with permit partial data leakage, enabling the creation of an amplification effect. This can be achieved by strategically selecting a subset of injected keywords. These injected keywords should be capable of forming unique combinations to identify the leaked documents effectively. Furthermore, these keywords have the potential to be recovered through the file injection and recovery process. After that, this subset is used to recover the leaked documents, subsequently facilitating the recovery of additional queries.

The crux of the matter lies in the selection of keywords for injection. To maximize the amplification effect, the instinct is to keep this set of injected keywords as concise as possible while ensuring their presence in a significant number of leaked documents, thereby enhancing the identification of more documents. Simultaneously, we aim to recover inert keywords. Consequently, we choose to pick several inert keywords from each leaked document to facilitate document identification.

The output of this attack is supposed to include both the injected inert keywords and the active keywords that are not part of the injected keyword universe.

The attack model is shown in Figure 4.

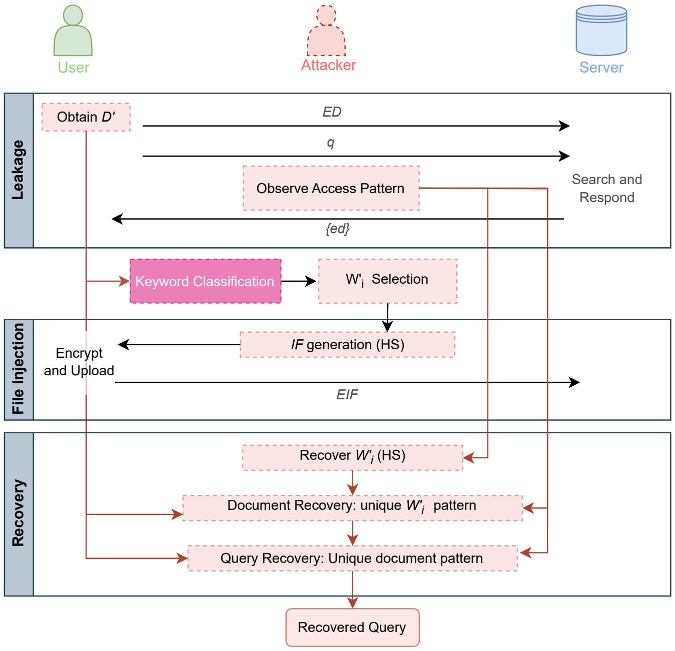


Fig. 4. Attack model in this work. The attacker has knowledge of the partially leaked data D' and the access patterns, and has the capability to actively generate and inject files into the system. The primary objective of the attack is to successfully recover the original queries, which we refer to as query recovery in the figure.

Leaked Knowledge. In our attack scenario, we assume that the attacker has gained knowledge of the leaked data and the access patterns. So the attacker knows 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 $m' \leq m$. The attacker is also aware of the number of keywords in the leaked documents $|d_i|$ and the number of queries in the encrypted documents $|ed_j|$.

C. Attack Procedure

The attack comprises three key procedure: keyword selection, file injection, and recovery. The objective is to recover inert keywords through file injection and subsequently utilize these recovered inert keywords for document recovery to uncover additional active keywords.

The overall procedure of the attack is outlined in Algorithm 1. We will now provide a step-by-step explanation for each procedure of the attack, together with corresponding detailed algorithms.

Algorithm 1 LEAP-Hierarchical Fusion Attack (LHFA)

```
Input: A set of known document $D'$ and corresponding set of known keyword $W'$
Output: A set of mapped query $R$
procedure INITIALIZATION($D'$, $W'$)
    $W'_{ac} \leftarrow$ KEYWORD_CLASSIFICATION($D', W'$)    ▷ Alg. 2
    return $W'_{ac}$
end procedure

procedure KEYWORD_SELECTION($D'$, $W'_{ac}$, $\alpha$)
    Initialize an empty list $W'_i$
    $W'_i \leftarrow$ KEYWORD_SELECTION($D', W'_{ac}, \alpha$)    ▷ Alg. 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
```

Initialization. After obtaining the leaked documents D', we proceed to extract the corresponding keyword set W'. The attacker then categorizes the extracted keywords into two distinct groups: active keywords, denoted as W'_{ac}, and local inert keywords, and local inert keywords that $k_i \in W'$ and $k_i \notin W'_{ac}$. Algorithm 2 outlines the procedure for distinguishing active keywords from W', and the local inert keywords are identified as the keywords in W' but not present in W'_{ac}.

Active keywords are those that exhibit unique patterns in D' and implies to also be active keywords in the entire dataset. On the other hand, keywords that do not display uniqueness within D' may still exhibit unique patterns in documents that were not leaked. Hence, these keywords are referred to as local inert keywords.

Algorithm 2 Keyword_Classification

```
Input: A set of known document $D'$ and corresponding set of known keyword $W'$
Output: A set of active keyword $W'_{ac}$
procedure KEYWORD_CLASSIFICATION($D'$, $W'$)
```

```
    Initialize an empty dictionary $Combination$ and an 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
```

Keyword Selection. Upon acquiring the set of active keywords W'_{ac}, Algorithm 3 is employed with W'_{ac} as input to determine the keywords selected for injection.

For each document, α keywords, specifically those not belonging to W'_{ac}—referred to as local inert keywords—are chosen to represent the document. This decision is motivated by the idea that the chosen keywords are meant to undergo the file injection black box process, thereby developing their unique access pattern through this procedure. Active keywords that already exhibit a unique pattern would undergo redundant computation, making it more effective to employ the file injection process exclusively for recovering keywords lacking distinct patterns—namely, the local inert keywords in this context.

If number of local inert keywords in a document is less than α, 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 $\alpha \cdot n'$.

Algorithm 3 Keyword_Selection

```
Input: A set of known document $D'$, a set of active keyword $W'_{ac}$ and a variable $\alpha$
Output: A set of chosen keyword for injection $W'_i$
procedure KEYWORD_SELECTION($D'$, $W'_{ac}$, $\alpha$)
    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
        $W'_i \leftarrow W'_i \cup k$
    end for
    return $W'_i$
end procedure
```

File Injection Black Box. We utilize the injection and recovery algorithm derived from the Hierarchical Search Attack (HSA) [11] as a black box for executing the injection step.

Given that HSA offers an advantage over Binary Search Attack (BSA) by addressing threshold countermeasures while maintaining a consistent 100% query recovery accuracy unaffected by other countermeasures, it stands as a preferable option.

With input the selected keywords W'_i that contains m'_i keywords and a maxi length of each injected file T, the injection algorithm return a set of files for injection IF, and the maximum size of IF is $\lceil \frac{m'_i}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$ by [11].

We encrypt the IF as EIF and inject it into the server. By observing the arrival time, we obtain the document matching C_i, which represents the match between the injected fils EIF and the server files IF. We then input C_i, W'_i, and the collected access patterns of ED and Q into the recovery black box.

The anticipated outcome of the entire black box is a recovered query set denoted as R_i, encompassing the mapping between injected keywords and their corresponding queries. The size of R_i is expected to be $|R_i| = |W'_i|$, aligning with the 100% accuracy asserted by HSA [11].

Document Recovery. In Algorithm 4, the recovered keyword set R_i is used to bootstrap the document recovery process. With input R_i we define each document d_i in D' as a set of queries $\{q_1, ..., q_a\}$, and then find the candidates for matching, for $q \in R_i$ and $a \geq \alpha$. The candidates are server documents ed_i that have the same set of queries as d_i. 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.

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 $C$
procedure DOCUMENT_RECOVERY
    Initialize an empty dictionary $C$
    for $ed_i \in ED$ do
        $ed_i \leftarrow \{q_1, ...q_a\}$, for $q \in R_i$
    end for
    for $d_i \in D'$ do
        $d_i \leftarrow \{q_1, ...q_b\}$, 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 $C$
end procedure
```

Remaining Keyword Recovery. In Algorithm 5, the utilization of the mapped document set C is directed towards the objective of matching additional keywords that are not part of the selected injected set W_i'. More precisely, the goal is to map all active keywords within the mapped documents by discerning their unique access patterns.

For each keyword in W' that is not in W_i', we define it based on a set of documents $\{d_1, ..., d_b\}$ where $d \in C$, and it is matched uniquely with a query that can be defined by the same set of documents. We merge the newly matched keywords with R_i to obtain the final set of mapped queries in the attack, denoted as R.

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 $R$
procedure REMAINING_KEYWORD_RECOVERY
    for $w_i \in W'$ do
        if $w_i \in R_i$ then
            remove $w_i$ from $W'$
        else
            $w_i \leftarrow \{d_1, ...d_\alpha\}$, for $d \in C$
        end if
    end for
    for $q_i \in Q$ and $q_i \notin R_i$ do
        if $q_i \in R_i$ then
            remove $q_i$ from $Q$
        else
            $q_i \leftarrow \{d_1, ...d_\alpha\}$, 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
    return $R$
end procedure
```

V. 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.

A. Setup

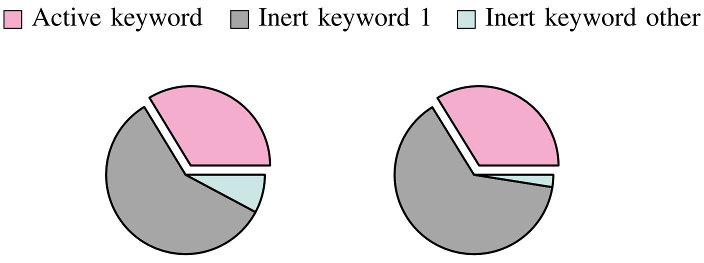
During our experiments, we evaluated the performance of the proposed attack by conducting tests on three real-life datasets: Enron [14], Lucene [15], and Wikipedia. These datasets were also used in the VAL attack [9]. Table III provides a summary of the datasets used, offering a brief overview of each.

TABLE III
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:** A publicly available collection of email communications generated by 150 senior managers of Enron Corporation. We specifically selected the emails from the `sent_mail` folder, which consists of 30,109 emails.
- **Lucene:** An email listing dataset capturing communication between users and PyLucene Developers. We focused on the "java-user" category, dedicated to addressing user issues. The chosen 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.
- **Wikipedia:** A widely recognized online encyclopedia created and maintained by a community of volunteers. We obtained a subset of 20,000 articles from a simple wiki dump provided by David Shapiro [16]. The dataset has a size of approximately 1.19 GB after decompression[1].

To preprocess the datasets, we removed common English stopwords using the NLTK package [17] in Python, and obtained the number of keywords that listed in Table III. This step helps filter out frequently occurring words that do not contribute significantly to the analysis.



(a) Lucene (b) Wiki(20000 files)

Fig. 5. Keyword Distribution of Dataset

We then analyzed the distribution of keywords in Lucene and Wikipedia as shown in Figure 5. Both datasets had a similar distribution, with approximately 33.7% active keywords and 66.3% inert keywords for Lucene, and approximately 33.8% active keywords and 66.2% inert keywords for Wikipedia. However, Lucene had a higher proportion (7.8%)

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

of inert keywords that appeared in more than one document compared to Wikipedia (2.5%). The keyword distribution of Enron is depicted in Figure 3.

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 α (2, 3, 5, and 10), and each experiment was repeated 10 times.

B. Evaluation Criterion

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

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

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

Query recovery accuracy is evaluated from two perspectives. The first aspect assesses how many queries have been correctly matched to their underlying keywords across the entire keyword universe. Given that our approach involves employing partial keywords to identify document identities via injection, which are subsequently recovered and utilized to initiate the recovery of remaining keywords within the recovered documents, this metric encompasses both queries recovered during the injection stage and those retrieved during the subsequent keyword recovery stage. The formula is outlined as follows:

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

The second aspect evaluates how many active keywords are correctly recovered among all active keywords in the keyword universe. This measure is represented as the ratio of correctly matched active keywords $R_a = \{k_a\}\ where\ k_a \in R\ and\ k_a \in W'_{ac}$ to all active keywords W'_{ac}:

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

In addition to document and query recovery accuracy, we introduce a criterion to analyze the amplification 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 follows:

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

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

C. Result and Comparison

The document and query recovery results for different datasets are presented in Figures 6, 7 and 8, respectively. Lines in a different color represent the results obtained with different α values, as indicated in the figures.

Document Recovery. Figure 6 shows that α values of 3, 5, and 10 achieve similar accuracy levels after a 1% data leakage for the Enron and Lucene datasets. In contrast, α 2 reaches a comparable accuracy after a 5% leakage. This suggests that for data leakages larger than 5% in Enron and Lucene, using α 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 α 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.

Query Recovery. In terms of query recovery, Figure 7 compares the recovered queries to the entire known keyword universe. It is observed that higher values of α lead to better results across all datasets. This is attributed to the fact that a larger set of injected keywords is obtained with higher α values, and a majority of these keywords are inert keywords that can only be recovered through the file injection step.

Figure 8 specifically focuses on the accuracy of recovering active keywords. It is notable that the Wikipedia dataset consistently outperforms the other datasets in this regard. When α is set to 10, Wikipedia achieves a perfect 100% accuracy in recovering active keywords, regardless of the leakage level.

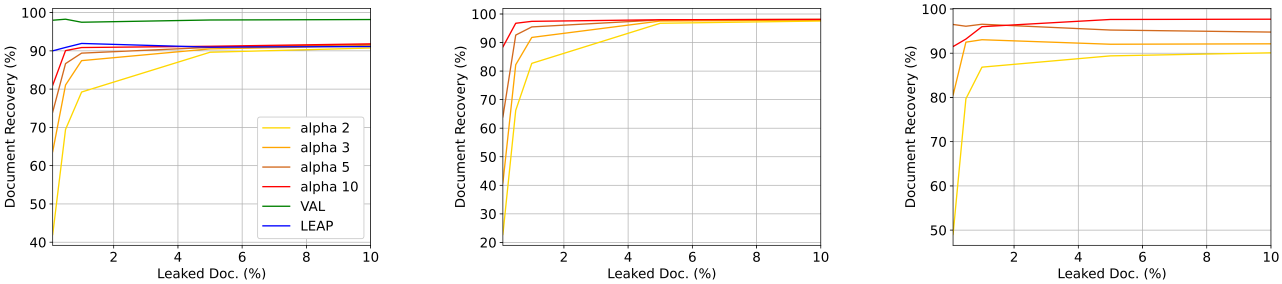
Table IV utilizes the rate of return as a metric to evaluate the amplification effect of the attack. For the specific numbers of injected and recovered keywords, please refer to Figure V in the Appendix.

On average, an α value of 2 tends to yield a relatively high rate of return, as shown in Table IV. However, an α value of 10 leads to the highest number of recovered queries, as indicated in Table V.

Once a 10% data leakage level is reached, the rate of return tends to stabilize. 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 suitable for Wikipedia, benefiting from its higher proportion of inert keywords in the dataset distribution.

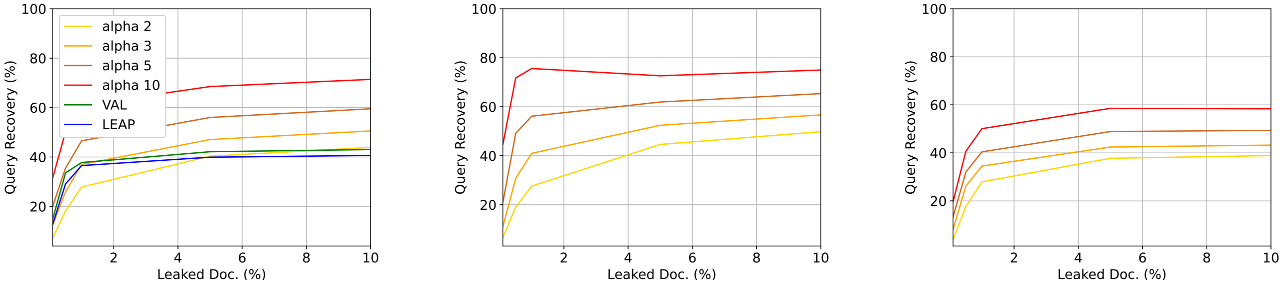
Comparison. As our attack combines the ideas from both inference attacks and file injection attacks, we compare our results with both types of SSE attacks.

In Figure 6a and 7a, we show the comparison with two inference attacks: LEAP [8] and VAL [9]. This limitation arises from the substantial increase in computation time observed with LEAP and VAL when utilizing the entire known keyword



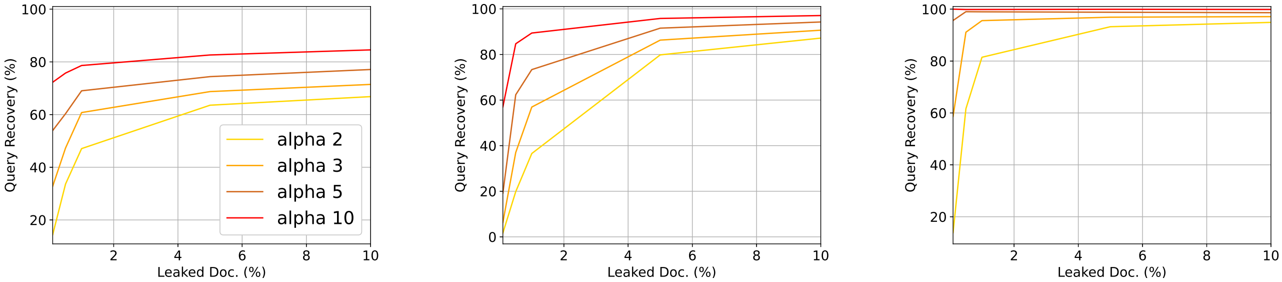
(a) Enron (b) Lucene (c) Wikipedia

Fig. 6. Document Recovery performance



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

Fig. 7. Query Recovery Performance Over the Entire Keyword Universe



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

Fig. 8. Query Recovery Performance Over Active Keywords

universe as input. The primary reason for this exponential growth in computation time is the recursive nature of computations performed by these methods for document and query recoveries. When only a few keywords are considered, the computational process remains manageable due to the limited number of combinations and iterations required. However, as the size of the input keyword universe expands, the number of possible combinations for document and query recovery increases exponentially. This is because each additional keyword introduces multiple new pathways for recursive computations, significantly escalating the overall computation time. Actually, within the timeframe of our project, Enron dataset, characterized by a smaller number of documents and keywords as outlined in Table III, was the least time-intensive among those considered. It highlights the practical challenges of using inference attacks like LEAP and VAL on large datasets with extensive keyword universes, where computation time can become a significant bottleneck.

In Figure 6a, all α 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.

In Figure 7a, our attack outperforms VAL and LEAP with α values of 5 and 10 for all leakage levels. Even with α

TABLE IV
RATE OF RETURN

Dataset Knowledge		Rate of Return (Avg.)		
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

set to 3, our attack surpasses both attacks after a leakage of approximately 1%, and even α 2 surpasses LEAP starting from a 5% leakage level, and exhibits a trend of surpassing VAL before reaching a 10% leakage level.

The comparison to file injection attacks, Hierarchical Search attack [11], and Decoding [6] is focus on the rate of return. For Hierarchical Search attack and Decoding, the highest rate of return is 1, indicating that all injected keywords lead to correct query recovery. Our attack, as shown in Table IV, achieves a rate of return greater than 1, even in the worst case. For example, in the Lucene dataset with an α value of 2, the rate of return is 1.088.

At the stable level, where the leakage level exceeds 10%, we have observed 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 $W_i' = \frac{|R|}{1.3}$.

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.

VI. Discussion

A. Correctness and Complexity Analysis

The proposed LHFA attack in Algorithm 1 consists of two main steps: 1) recovering inert keywords through file injection (lines 1-14), and 2) uncovering additional active keywords through document recovery (lines 15-20).

Complexity. The file injection results in a complexity of $\lceil \frac{m_i'}{2T} \rceil \cdot (\lceil \log 2T \rceil + 1) - 1$, i.e., the number of files for injection, as discussed above. The term m_i' represents the number of inert keywords, with a maximum size of $\alpha \cdot n'$, since we select α inert keywords from each of the known document set D', as shown in Algorithm 3. Therefore, the complexity for file injection is $O(\alpha \cdot n')$ for a constant parameter T.

Subsequently, the recovered inert keyword enables document recovery in Algorithm 4. The major computation comes from comparing $|ED| \times |D'|$ vectors of length at most m', which therefore has a complexity of $O(n'nm')$. The same applies to the following active keyword recovery in Algorithm 5, which compares at most $|W| \times |Q|$ vectors of length $|C|$ and has complexity $O(m'mn')$. Therefore, the total complexity for recovering active keywords is $O(m'n'(m+n))$.

Correctness. According to [6], [10], [11], the file injection attack in the first step achieves a 100% accuracy in recovering inert keywords. For the active keywords, the recovery accuracy relies heavily on the distribution of keywords, and only experimental results are demonstrated in previous works LEAP and VAL, on which our attack is built. As shown in Figures 6a and 7a, our attack exhibits higher query recovery accuracy than previous works.

B. Countermeasures

While it is indeed gratifying to acknowledge that the LEAP-Hierarchical Fusion Attack (LHFA) attack has successfully achieved our initial objective of creating an amplification effect for file injection attacks and has made strides in addressing the inherent limitations of inference attacks, it remains paramount, from a cybersecurity perspective, to engage in an ongoing discourse concerning countermeasures that can effectively thwart our newly proposed attack.

Given the innovative nature of our LHFA, which seamlessly amalgamates the strengths of both inference attacks (passive) and file injection attacks (active), we will delve into the implications of various access pattern hiding countermeasures on the LHFA attack.

Obfuscation and padding, as detailed in [18], [19], [20], [21], serve as initial deterrents by returning bogus encrypted documents to queries. However, the operational intricacy lies in effectively implementing these countermeasures. Simple static padding and obfuscation [20], [19] might seem a viable route, but in practice, they only offer limited resistance against our attack. While they can distort some relations at the setup stage, our method's adaptive nature still facilitates document recovery since the observed access pattern of injected files remains unaffected. The challenge heightens when considering document matching based on unique query combinations $(q_i, ...q_a)$, as the use of padding and obfuscation can also inadvertently decrease the effectiveness of genuine query processes.

TABLE V
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

Dynamic padding, as introduced in [21], might appear to be a more rigorous countermeasure, given its continuous update of the bogus access pattern. However, its real-time effectiveness against LHFA remains an area of intricate exploration. The continuous alterations might obstruct query recovery during the file injection phase, but it also introduces computational overhead and complexity.

The countermeasures such as Vaccine technique [22], while being specifically crafted for resisting file injection attacks, have their own set of challenges. It necessitates the creation of a self-injected file that mirrors the actual injected file in terms of keywords and document size. Maintaining such intricate correlations while ensuring server efficiency is non-trivial. Even if successfully applied, the precision decrease in injected query recovery remains a concern. Thus, while the Vaccine technique does pose a challenge, the robustness and adaptability of LHFA ensure its significance in the field.

VII. CONCLUSION

Our study confirms the effectiveness of file injection attacks in achieving an amplification effect when leaked documents are available, aligning with the “first document recovery, then query recovery” concept proposed in the LEAP attack [8]. Furthermore, comparing our query recovery results to the LEAP and VAL approaches highlights the effectiveness of file injection in mitigating the confusion caused by inert keywords in inference attacks. The experimental results have yielded a significant insight: datasets characterized by a higher proportion of inert keywords prove to be more conducive targets for our attack. This observation underscores the importance of factoring in dataset types or distributions when planning future attacks based on the LHFA approach.

In future research, we plan to incorporate additional leakage patterns into our current attack to explore the potential for achieving higher accuracy in document recovery, while further amplifying the number of recovered queries. Although our attack does not directly counter existing countermeasures, our suggested keyword classification can serve a dual purpose: aiding in circumventing countermeasures and facilitating the development of more effective countermeasures. Regarding the design of future SSE attacks, our research highlights that existing inference attacks are already quite capable of recovering active keywords. Consequently, the focus should shift towards distinguishing inert keywords more effectively. From the perspective of bolstering SSE security, acknowledging inert keywords as inherent thresholds for access pattern inference attacks, countermeasures can prioritize safeguarding active keywords. This optimization can lead to more efficient cache utilization and a reduced countermeasure workload.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments. This work was partly supported by the EU Horizon Europe Research and Innovation Programme under Grant No. 101021727 (IRIS), No. 101073920 (TENSOR), No. 101070627 (REWIRE), and No. 101070052 (TANGO).

References

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*. IEEE, 2000, pp. 44–55.

[2] H. Zhou, Z. Shi, X. Ouyang, and Z. Zhao, "Building a blockchain-based decentralized ecosystem for cloud and edge computing: an allstar approach and empirical study," *Peer-to-Peer Networking and Applications*, vol. 14, no. 6, pp. 3578–3594, 2021.

[3] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: ramification, attack and mitigation." in *NDSS*, vol. 20. IEEE, 2012, p. 12.

[4] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 668–679.

[5] D. Pouliot and C. V. Wright, "The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1341–1352.

[6] L. Blackstone, S. Kamara, and T. Moataz, "Revisiting leakage abuse attacks," *Cryptology ePrint Archive*, 2019.

[7] J. Ning, J. Xu, K. Liang, F. Zhang, and E.-C. Chang, "Passive attacks against searchable encryption," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 789–802, 2018.

[8] J. Ning, X. Huang, G. S. Poh, J. Yuan, Y. Li, J. Weng, and R. H. Deng, "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*, 2021, pp. 2307–2320.

[9] S. Lambregts, "Revisit attacks on searchable symmetric encryption: Explore more, reveal more," Master's thesis, Delft University of Technology, 2022.

[10] X. Zhang, W. Wang, P. Xu, L. T. Yang, and K. Liang, "High recovery with fewer injections: Practical binary volumetric injection attacks against dynamic searchable encryption," *arXiv preprint arXiv:2302.05628*, 2023.

[11] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption." in *USENIX Security Symposium*, vol. 2016, 2016, pp. 707–720.

[12] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," *Cryptology ePrint Archive*, 2014.

[13] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 639–654.

[14] W. W. C. MLD, "Enron email datasets," 2015, accessed: Jun. 01, 2023. [Online]. Available: https://www.cs.cmu.edu/∼enron/

[15] T. A. S. Foundation, "Apache: Mail archieves of lucene," 1999, accessed: Jun. 01, 2023. [Online]. Available: https://lists.apache.org/\#lucene

[16] D. Shapiro, "Plaintextwikipedia: Convert wikipedia database dumps into plaintext files," 2021, accessed: Jun. 01, 2023. [Online]. Available: https://github.com/daveshap/PlainTextWikipedia

[17] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[18] Y. Kortekaas, "Access pattern hiding aggregation over encrypted databases," Master's thesis, University of Twente, 2020.

[19] Z. Shang, S. Oya, A. Peter, and F. Kerschbaum, "Obfuscated access and search patterns in searchable encryption," *arXiv preprint arXiv:2102.09651*, 2021.

[20] G. Chen, T.-H. Lai, M. K. Reiter, and Y. Zhang, "Differentially private access patterns for searchable symmetric encryption," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 810–818.

[21] R. Du, Y. Zhang, and M. Li, "Database padding for dynamic symmetric searchable encryption," *Security and Communication Networks*, vol. 2021, pp. 1–12, 2021.

[22] H. Liu, B. Wang, N. Niu, S. Wilson, and X. Wei, "Vaccine:: Obfuscating access pattern against file-injection attacks," in *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2019, pp. 1–9.