Query Recovery from Easy to Hard
Jigsaw Attack against SSE

Nie, Hao; Wang, Wei; Xu, Peng; Zhang, Xianglong; Yang, Laurence T.; Liang, Kaitai

Publication date
2024

Document Version
Final published version

Published in
Proceedings of the 33rd USENIX Security Symposium

Citation (APA)
Nie, H., Wang, W., Xu, P., Zhang, X., Yang, L. T., & Liang, K. (2024). Query Recovery from Easy to Hard: Jigsaw Attack against SSE. In *Proceedings of the 33rd USENIX Security Symposium* (pp. 2599-2616). (Proceedings of the 33rd USENIX Security Symposium). USENIX Association.

Query Recovery from Easy to Hard: Jigsaw Attack against SSE

Hao Nie and Wei Wang, *Huazhong University of Science and Technology;* Peng Xu, *Huazhong University of Science and Technology, Hubei Key Laboratory of Distributed System Security, School of Cyber Science and Engineering, JinYinHu Laboratory, and State Key Laboratory of Cryptology;* Xianglong Zhang, *Huazhong University of Science and Technology;* Laurence T. Yang, *Huazhong University of Science and Technology and St. Francis Xavier University;* Kaitai Liang, *Delft University of Technology*

https://www.usenix.org/conference/usenixsecurity24/presentation/nie

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Query Recovery from Easy to Hard: Jigsaw Attack against SSE

Hao Nie[1], Wei Wang[1,✉], Peng Xu[1,2,3,4,✉], Xianglong Zhang[1], Laurence T. Yang[1,5], and Kaitai Liang[6]

[1] *Huazhong University of Science and Technology*
[2] *Hubei Key Laboratory of Distributed System Security, School of Cyber Science and Engineering*
[3] *JinYinHu Laboratory*
[4] *State Key Laboratory of Cryptology*
[5] *St. Francis Xavier University*
[6] *Delft University of Technology*
✉ *Corresponding authors: viviawangwei@hust.edu.cn, xupeng@hust.edu.cn*

Abstract

Searchable symmetric encryption schemes often unintentionally disclose certain sensitive information, such as access, volume, and search patterns. Attackers can exploit such leakages and other available knowledge related to the user's database to recover queries. We find that the effectiveness of query recovery attacks depends on the volume/frequency distribution of keywords. Queries containing keywords with high volumes/frequencies are more susceptible to recovery, even when countermeasures are implemented. Attackers can also effectively leverage these "special" queries to recover all others.

By exploiting the above finding, we propose a Jigsaw attack that begins by accurately identifying and recovering those distinctive queries. Leveraging the volume, frequency, and co-occurrence information, our attack achieves 90% accuracy in three tested datasets, which is comparable to previous attacks (Oya et al., USENIX' 22 and Damie et al., USENIX' 21). With the same runtime, our attack demonstrates an advantage over the attack proposed by Oya et al (approximately 15% more accuracy when the keyword universe size is 15k). Furthermore, our proposed attack outperforms existing attacks against widely studied countermeasures, achieving roughly 60% and 85% accuracy against the padding and the obfuscation, respectively. In this context, with a large keyword universe ($\geq$3k), it surpasses current state-of-the-art attacks by more than 20%.

1 Introduction

Searchable Symmetric Encryption (SSE) [2, 4, 7, 8, 12, 23, 33, 38, 41, 48] enables users to securely search encrypted databases stored on remote servers. An SSE scheme typically consists of setup and search protocols. In the setup, the user sends encrypted indexes of the documents to the server. In the search, the user generates a search token and sends it to the server who then returns the matched documents. The search process does not reveal any confidential information about the documents or the user's search query, except for the volume pattern (also known as the response length) and the access pattern, which reveal the number of matched documents and their identities, respectively. Additionally, the server may also know the search pattern, which indicates whether two queries are identical by comparing their search tokens or access patterns.

Passive attacks on SSE can exploit the above leakages and some prior knowledge to recover queries. According to the prior knowledge given to the attacker, we categorize two main attacks in Table 1: 1) **known-data attacks** [1, 5, 22, 25, 30, 35, 46], which assume that the attacker has access to partial/full plain texts of the documents in the user's dataset; and 2) **similar-data attacks** [13, 26, 31, 32, 35], which enable the attacker to obtain a similar document set or estimations on the users' query distribution. Unlike known-data attacks, that require the plain texts, similar data attacks can recover queries by exploiting statistical information from a similar dataset, such as query frequency and the probability of two keywords appearing in the same document. Without relying on the "strong assumption" that the attacker must be provided plain texts of documents, similar-data attacks are relatively practical to deploy and bypass countermeasures. Existing works [31, 32] have shown that similar-data attacks can bypass some defenses employing pattern randomization techniques [10, 36]. We note that SSE is also vulnerable to active attacks [1, 34, 50, 51] leveraging file injection to recover queries, which is orthogonal to this work.

Damie et al. [13] explored an intriguing phenomenon across various query distributions, indicating the correlation between accuracy and query volume (i.e., the number of documents containing a particular keyword). Oya et al. [31] demonstrated that high-frequency queries (i.e., the frequency with which the user queries a specific keyword) give a greater probability of being successfully recovered in their proposed attack. We note that a similar notice was given in [1] that the effectiveness of known-data attacks is also influenced by the volume of queries, wherein high-volume queries are easily recoverable. Building upon the aforementioned interesting

Table 1: Comparisons of existing passive attacks[1].

Attack	Leakage	Known prior knowledge		Similar prior knowledge		Accuracy	Padding[2]	Obfuscation[3]
		Document	Query	Document	Frequency			
IKK [22]	ap	●	◐	○	○	$\sim 80\%$	-	-
Count [5]	ap, vp	●	○	○	○	$\sim 90\%$	-	-
SubgraphID [1]	ap	◐	○	○	○	$\sim 90\%$	-	-
LEAP [30]	ap	◐	○	○	○	$\sim 100\%$	-	-
RSA [13]	ap	○	◐	●	○	$\sim 85\%$	$< 20\%$	$< 20\%$
Freq [26]	sp	○	○	○	●	$\sim 20\%$	$\sim 20\%$	$\sim 20\%$
SAP [31]	vp,sp	○	○	●	●	$\sim 50\%$	$\sim 30\%$	$\sim 30\%$
GraphM [35]	ap	○	○	●	○	$\sim 70\%$	$< 20\%$	$< 20\%$
IHOP [32]	sp,ap	○	○	●	●	$\sim 90\%$	$< 20\%$	$\sim 85\%$
Jigsaw (Ours)[4]	vp,sp,ap	○	○	●	◐	$\sim 90\%$	$\sim 60\%$	$\sim 85\%$

[1] "ap" denotes the access pattern, "vp" denotes the volume pattern, and "sp" denotes the search pattern. The "●" indicates that the attack needs nearly all known data or strongly relies on the corresponding similar data. The "◐" indicates that the attack needs partial known data or not particularly relies on similar data. The "○" means that the attack does not need any known data or similar data. The first four attacks are known-data attacks, and the last five are similar-data attacks. The RSA mainly relies on similar data but needs a few known queries to start the attack. We do not present the performance of the first four attacks against padding and obfuscation (denoting "-") since we mainly focus on similar-data attacks.

[2] The performance against the padding of CGPR [5] with $k = 1{,}000$ on Enron. We utilize an adaptation (Appendix B) for Jigsaw. Padding is claimed to mitigate the listed known-data attacks effectively [1,30].

[3] The performance against the obfuscation of CLRZ [10] with TPR= 0.999, FPR= 0.05 on Enron. Note adaptations (Appendix B) are used for IHOP and Jigsaw.

[4] Our attack can reach 90% accuracy even if a defense hides the sp, and thus one may consider the sp an optional attack advantage.

hints, we conduct experiments that confirm the influence of volume and/or frequency on the performance of attacks. Moreover, we discover that leveraging this knowledge enables us to enhance the effectiveness of similar-data attacks. We provide two crucial observations regarding the query recovery.

Observation 1. *Queries containing keywords with a high volume/frequency are much easier to recover than others.*

In a database, the volume of keywords follows Zipf's law [52], which states that the volume of a keyword ranks nth in a sorted list (sorted by volume) is inversely proportional to n. We also observe that the frequency of keywords follows almost the same law. We confirm the above phenomenons by showing the concrete results in three datasets (See Section 4.1 and Appendix A for more details). Keywords with higher volume or frequency display larger disparities, which consequently makes it easier for attackers to recover those queries.

Observation 2. *By revealing the queries from observation 1, the attacker can gain advantage to retrieve further queries (even all queries).*

In [13], Damie et al. proposed an efficient similar-data attack (i.e. refined score attack, RSA) that achieves around 85% accuracy in recovering all queries by utilizing only 10 known queries. They also show that when utilizing known queries with a higher volume, the attack's accuracy increases and becomes more stable.

Challenges. Observation 1 does not explicitly facilitate a way to identify and recover those distinctive queries as they consistently intermingle with others. To the best of our knowledge, there is no attack that first focuses on filtering those distinctive queries, thereby allowing for the recovery of queries from easy to hard. Setting a start with immediately recovering all queries based on known or recovered queries is not trivial and could be defended against by countermeasures. For example, previous work [13] cannot work effectively at the outset without any proper known query set. This also implicitly explains why it achieves low accuracy under padding [5] and obfuscation [10].

Contributions. We propose a new effective similar-data attack called Jigsaw providing a "granular and incremental" strategy, which comprises three core modules. 1) The first module uses the keyword's volume and frequency information to locate and recover the most distinctive queries. 2) The second module further refines the recovered queries by matching the queries with the keywords according to the co-occurrence matrix. This module eliminates those incorrect query recoveries from the first module and tries to achieve near-perfect accuracy. 3) The last module is to recover the remaining queries using the outputs from the second module. We generate scores for query-keyword combinations using the co-occurrence matrix, volume, and frequency information. We optimize the score for queries to obtain matches between queries and keywords. We also provide comprehensive evaluations of Jigsaw. Concretely, our contributions are outlined as follows.

- *Localization and recovery of distinctive queries.* We measure the distinguishability of each query and use the volume and frequency to recover the most distinguishable queries (the first module of Jigsaw). For queries with a high volume and frequency (e.g., the top 10% of queries in volume and frequency in Enron [44], Lucene [15], and Wikipedia [16]), we can obtain an accuracy $> 70\%$.

• *Precise verification of recovered queries.* We make a further refinement by filtering out those queries that do not align well with the co-occurrence information (the second module). We here obtain nearly 100% accuracy at the expense of recovering a smaller number of queries (about a dozen queries in Enron and 50 queries in Lucene and Wikipedia).

• *Accurate recovery of all queries.* We at last utilize the recovered queries to recover all queries with about 95% accuracy (the last module). Even if the frequency information is not given, we can still capture roughly 90% accuracy. We state that Jigsaw exhibits durability, as it can hold its effectiveness (dropping $< 5\%$ of accuracy) in the future period even by exploiting the auxiliary frequency information that was leaked long ago (e.g., 30 months in Wikipedia and 150 weeks in Enron and Lucene).

• *Comprehensive evaluations and comparisons.* We present empirical experiments and comparisons with the state of art similar-data attacks (including Graphm [35], SAP [31], RSA [13], and IHOP [32]) to highlight the performance of Jigsaw. Our attack provides $> 90\%$ accuracy surpassing the Graphm and SAP attacks, similar to the RSA and IHOP. Within the same runtime, Jigsaw exhibits about 15% more accuracy than IHOP when the keyword universe is 15k. Also, Jigsaw outperforms them when countering the defenses (padding [5] and obfuscation [10]). It maintains $> 60\%$ and $> 85\%$ accuracy against the padding (in most cases) and obfuscation and takes the lead in accuracy in most cases.

2 Related Work

Except for the SSE schemes [14, 17, 45] based on expensive primitives, such as ORAM and PIR, most SSE schemes leak the access pattern, search, volume, and response size pattern (i.e., the size of each document). With some prior knowledge, passive attacks abuse the above leakages to recover users' queries. These attacks can be categorized as similar-data attacks [13, 26, 31, 32, 35] and known-data attacks [1, 5, 22, 25, 30, 35, 46].

Similar-data attacks. Liu et al. [26] proposed the Freq attack that exploits the search pattern and the query frequency information to recover users' queries. Recently, Oya et al. [31] proposed the SAP attack, which utilizes the search and volume pattern to get the frequency and volume of each query. However, Freq and SAP strongly rely on the frequency information and achieve a relatively low accuracy. Attacks that abuse the access pattern have higher accuracy. Pouliot et al. [35] proposed the GraphM attack that formalizes the query recovery as a weighted graph match problem and solves it by PATH [49] or Umeyama [42] algorithm. Oya et al. [32] proposed the IHOP attack, which uses a co-occurrence matrix of queries and keywords, along with query frequency, to launch its attack. IHOP supposes that queries are correlated and follow a Markov process, allowing it even to threaten frequency-smoothing defenses such as PANCAKE [20]. Damie et al. [13] proposed the RSA, which starts with some known queries. Though providing an accuracy of about 85%, RSA still requires some known queries as a prerequisite. Without those, the attack will not work effectively.

Known-data attacks. Islam et al. [22] proposed the first known-data attack (IKK) with all documents and partial queries to recover queries. Cash et al. [5] proposed the Count attack that can recover most queries without known queries. In [1], Blackstone et al. proposed an attack that performs perfectly (accuracy approaching 100%) with fully known documents, but poorly (less than 10%) with a small portion of known documents. Ning et al. [30] proposed LEAP, which can recover half of all queries with 100% accuracy with only 1% of documents. These attacks are all dependent on known data information. However, the known data are hard to obtain, and the attacks are easy to prevent with countermeasures.

Countermeasures. To defend against leakage abuse attacks, many countermeasures [3, 5, 10, 14, 22, 43, 47] have been proposed. Among those, padding is one of the most commonly used methods. Cash et al. [5] first presented the padding strategy. The volume of each query is padded to the nearest multiple of an integer k. After that, Demertzis et al. [14] proposed SEAL, which pads the volume of each query to the nearest power of an integer x. The padded documents would add noise to the volume and access patterns, hampering attacks abusing those leakages. Several works also consider the keywords clustering [3, 43]. Each cluster contains no less than α keywords, and then each keyword is padded to the largest volume in the cluster.

Another countermeasure is obfuscation [10]. When querying for a keyword, if a document contains the keyword, the document will be returned with probability p (the true positive rate, TPR); otherwise, each document will be returned with probability q (the false positive rate, FPR). Shang et al. [36] proposed OSSE, which provides the same response effect and produces fresh obfuscation in each query.

3 SSE Scheme and Attack Model

We revisit the standard SSE [12], define the leakage function of queries, and describe the attacker's prior knowledge as the prerequisite of our attack. We put the summary of notations in the full version [29].

3.1 SSE

An SSE scheme [12] facilitates keyword searches over encrypted data, denoted as ED, while maintaining the confidentiality of the data and the keywords. The typical components of an SSE scheme encompass the setup, update, and query processes. Initially, the user possesses a dataset D, which comprises a set of documents d identified by $id(d)$. Each document contains a list of keywords k. During the setup phase, the user can construct and encrypt an index, and then upload it along with the encrypted document set ED to the server. In the update, the user can dynamically update the index stored on the server. During the query process, to search a keyword k, the user generates a trapdoor $td(k)$ for the server. Eventually, the user retrieves the list $D(k)$, which is a list of $id(d)$ satisfying that k appears in d. We also denote the list $D(k)$ as $D(td(k))$ for convenience. We note that the user also needs to retrieve the encrypted documents according to the $id(d)$ and decrypt them to complete the search.

3.2 Leakages

An efficient SSE scheme typically leaks the volume pattern, the access pattern, and the search pattern of queries to the server and potential eavesdroppers. For a sequence of s queries $Td^s = [td(x_1), td(x_2), \ldots, td(x_s)]$, the leakages often used in attacks are summarized as follows.

- **Access pattern** is the family of functions $ap : ED \times Td^s \to AP^s$ where AP is a list $[D(x_1), D(x_2), \ldots, D(x_s)]$. For each query, the scheme leaks the identifiers of corresponding encrypted documents. This leakage happens in most SSE schemes [2, 4, 6, 9, 12, 24, 28, 39, 48] when the user retrieves the encrypted document. Some schemes use primitives such as ORAM [18] or PIR [11] to hide access pattern, but those primitives lead to expensive costs.
- **Volume pattern** is the family of functions $vp : ED \times Td^s \to VP^s$, where VP is a list $[|D(x_1)|, |D(x_2)|, \ldots, |D(x_s)|]$. For each query, the scheme leaks the number of documents returned by the server.
- **Search pattern** is the family of functions $sp : ED \times Td^s \to M^{s\times s}$, where $M^{s\times s}$ is a $s \times s$ binary matrix such that $M^{s\times s}[i, j] = 1$, if the underlying keywords of td_i and td_j are the same and otherwise $M^{s\times s}[i, j] = 0$. For any two queries $td(x_i), td(x_j) \in Td, i \neq j$, the attacker knows whether x_i equals x_j. While schemes may not directly reveal the search pattern from queries, repeated querying of the same keyword leads to the exposure of the same access pattern. Attackers can utilize this access pattern to infer whether two queries correspond to the same keyword [31].

3.3 Attackers

We respectively consider two kinds of attackers targeting the user's queries: honest but curious servers and eavesdroppers:

- An *honest but curious server* follows the SSE protocols but attempts to recover the users' queries by utilizing the leakage pattern and other prior knowledge, such as known data or similar data. The server also has access to all the encrypted documents.
- An *eavesdropper* who intercepts the traffic between the server and the user can observe the encrypted documents returned in each query and possess the same knowledge of leakages as the server, except for all the encrypted documents. With a similar dataset, the eavesdropper can also launch similar-data attacks.

Different from known-data attacks, the server could utilize outdated (obtained from the past) or leaked documents which are not necessarily included in the current user's dataset to launch similar-data attacks. Note the eavesdropper could also know some similar data in several scenarios by acting as a legal user. For instance, the eavesdropper may share the same email system (producing similar email data) with his colleagues. We state that both attackers can employ our Jigsaw attack by utilizing the following knowledge derived from leakage and similar data.

Attackers' knowledge derived from leakages. We use the leakages to derive the frequency, volume, and co-occurrence of queries. We assume the user issues s queries, denoted as Td^s, from which the attacker identifies l different queries by the search pattern. We denote the query list identified by the attacker as $Td_r = [td_1, td_2, \ldots, td_l]$, which does not contain repeated queries. The attacker can also observe the returned documents $[D(td_1), D(td_2), \ldots, D(td_l)]$. The server possesses knowledge of the total number of documents, denoted as $|D|$. While the eavesdropper, who can only observe the query traffic, does not know the total number of documents. We also use $|D|$ to represent the size of the document set observed by the eavesdropper. Then for each query td, we normalize the volume pattern of query denoted $v_{td} = |D(td)|/|D|$. For the query list Td_r, $V_r = [v_{td_1}, v_{td_2}, \ldots, v_{td_l}]$ is the vector of volume of all queries in Td_r. With the search pattern, the attacker acquires knowledge of the frequency at which a td appears in Td^s. We denote the frequency of td as $f_{td} = Count(td)/|Td^s|$, where $Count(td)$ computes the number of td in Td^s. For the query list Td_r, $F_r = [f_{td_1}, f_{td_2}, \ldots, f_{td_l}]$ is the vector of frequency of all queries in Td_r. Based on the access pattern, the attacker can construct a $l \times |D|$ matrix ID_r. $ID_r[i, j] = 1$, if the search result for td_i contains d_j, 0 otherwise. Then, the attacker can construct the $l \times l$ co-occurrence matrix $C_r = ID_r ID_r^\top / |D|$.

Attackers' knowledge derived from similar data. We assume the attacker knows a list of similar documents $D_s = [d_1, d_2, \ldots, d_k]$ and employs the same algorithm to extract keywords as the user. Actually, the attacker can easily obtain a similar keyword universe as the user's database with a small similar data [50]. We denote the keyword universe extracted by the attacker as $W_s = [w_1, w_2, \ldots, w_m]$. Then, the attacker can construct the volume $V_s = [v_{w_1}, v_{w_2}, \ldots, v_{w_m}]$, where $v_{w_i} = |D_s(w_i)|/|D_s|$ and the $D_s(w_i)$ is the documents in D_s which contain keyword w_i. The attacker also constructs ID_s from W_s and D_s in a similar manner as the construction of ID_r. Based on the ID_s, the attacker calculates the co-occurrence matrix $C_s = ID_s ID_s^\top / |D_s|$. The attacker can also obtain a similar query frequency $F_s = [f_{w_1}, f_{w_2}, \ldots, f_{w_m}]$ for W_s by public information such as Google Trend [19] or

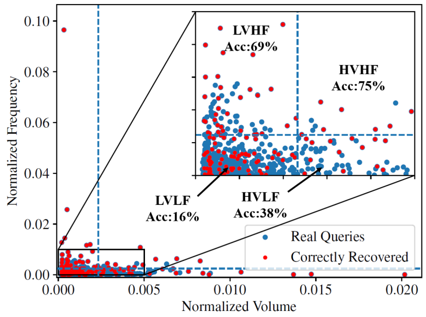


Figure 1: The distribution of queries on Enron. The horizontal dashed line divides the top 10% queries on volume from other queries; the vertical dashed line divides the top 10% queries on frequency from other queries. The blue dots denote the real queries issued by the user; the red dots denote the queries successfully recovered by the simple attack presented in Appendix A.

outdated frequency information.

4 Jigsaw Attack

We first demonstrate that Observation 1 does apply to all tested datasets although it does not directly provide an approach to recover those distinctive queries. We introduce the differential distance in the first module of Jigsaw to establish a way to distinguish these queries. We then define a distance between queries and keywords, enabling us to pair the distinctive queries with their nearest keywords. We accomplish the refinement in the second module by testing the queries' compatibility with co-occurrence relationships. Building on Observation 2, in the third module, we systematically recover all remaining queries by using those obtained in the second module. We define a specified "score" for each query-keyword pair and maximize it to get matches. The recovery process is iterative, with previously recovered queries being utilized in the recovery of subsequent queries.

4.1 Distribution of Keywords

Before proceeding to the Jigsaw attack, we present a comprehensive exposition of volume and frequency distribution (commonly found) in typical databases. We also showcase how the distribution impacts the effectiveness of attacks.

According to the Zipf's law [52], the volume of keywords in a database follows the Zipfian distribution, illustrating that *a small portion of keywords associates with a high volume.* As a result, these keywords exhibit significant disparities among each other due to their limited quantity but extensive volume range. A similar phenomenon emerges concerning query frequency, which indicates that a small subset of keywords could tend to be more frequently queried by users.

We present a simple similar-data attack to show that queries with high volume and frequency are relatively easier to recover. We employ this attack solely to demonstrate the distribution of keywords and its influence on the recovery. In this attack, the attacker generates the volume and frequency of queries and pairs them with keywords that exhibit the closest similarity w.r.t. volume and frequency. We simulate the attack on Enron [44], Lucene [15], and Wikipedia [16] and provide the results for Enron in Figure 1 (See Appendix A for details regarding the attack and its results on other datasets).

Figure 1 shows the volume and frequency correlated distribution of queries and the effectiveness of the attack (on recovery). We select the top 1, 000 keywords on volume except for the stop words as the keyword universe and categorize the corresponding queries into four quadrants: HVHF, HVLF, LVHF, and LVLF, where "L" and "H" represent "low" and "high", and the "V" and "F" denote "volume" and "frequency", respectively. High-volume queries refer to the top 10% of queries based on volume, while low-volume queries are the remaining 90%. The same classification applies to high-frequency and low-frequency queries. The lower-left corner of the figure is magnified to highlight that keywords with low volume and low frequency exhibit dense packing, which makes it difficult to differentiate these keywords. In contrast, the HVHF quadrant consists of sparser queries that are more distinguishable from each other. Using the similarity in volume and frequency, the attacker can easily recover these queries. Experimental results confirm this, with the simple attack achieving an accuracy of nearly 80% in the HVHF quadrant ($>$ 90% in Lucene and Wikipedia datasets). In the LVLF quadrant, however, the accuracy drops to 10%. We also achieve a moderately high accuracy of 62% and 34% in the LVHF and HVLF quadrants. Similar findings in [1, 13, 31] also present the correlation between frequency, volume, and accuracy.

4.2 Locating and Recovering the Most Distinctive Queries

As previously described, some queries are more distinguishable and easier to recover. Our first module aims to locate and recover these queries. We outline its details in Algorithm 1, which takes the Td_r, V_r, and F_r derived from query leakages in SSE and W_s, V_s, and F_s from similar data as input and outputs *BaseRec* predictions *Pred*. The *BaseRec* determines the number of recovered queries in this module. As we recover queries from high to low distinctiveness, a larger value of *BaseRec* can yield more predictions of queries lacking distinctiveness, consequently leading to a decline in accuracy.

Concretely, we first identify the most distinctive queries from all the queries by evaluating the *differential distance*. We define the differential distance d_{td_i} of a query td_i as follows:

$$d_{td_i} = \min_{td_j \in Td_r \wedge j \neq i} \alpha \cdot |v_{td_i} - v_{td_j}| + (1-\alpha)|f_{td_i} - f_{td_j}|. \tag{1}$$

Algorithm 1: Recover the top-*BaseRec* distinctive queries.

```
1  procedure RecoverDQ($Td_r, V_r, F_r, W_s, V_s, F_s, \alpha, BaseRec$)
2      $Dis \leftarrow \emptyset$; ▷ $Dis$ maintains the differential distances of queries;
3      for all $td_i \in Td_r$ do
4          $d_{td_i} = \min_{td_j \in Td_r \wedge j \neq i} \alpha \cdot |v_{td_i} - v_{td_j}| + (1-\alpha)|f_{td_i} - f_{td_j}|$;
5          append $(td_i, d_{td_i})$ to $Dis$;
6      end
7      Sort $Dis$ in descending order according to $Dis.d_{td}$;
8      $Pred \leftarrow \emptyset$; ▷ $Pred$ stores the recoveries of distinctive queries;
9      for $i \in [BaseRec]$ do
10         $td_i, d_{td_i} = Dis[i]$;
11         $w = \arg\min_{w_j \in W_s} \alpha \cdot |v_{td_i} - v_{w_j}| + (1-\alpha)|f_{td_i} - f_{w_j}|$;
12         append $(td_i, w)$ to $Pred$;
13     end
14     return $Pred$;
15 end
```

Algorithm 2: Verification by co-occurrence matrices.

```
procedure Verify($Pred, C'_r, C'_s, BaseRec, ConfRec$)
    $Temp\_Pred \leftarrow Pred$;
    $Td' \leftarrow Pred.td$;
    $Revconf \leftarrow \emptyset$;
    for $i \in [|Td'|]$ do
        $revconf = ||C'_r[i] - C'_s[i]||$;
        append $(Td'[i], revconf)$ to $Revconf$;
    end
    Sort $Revconf$ in descending order according to the $Revconf.revconf$;
    for $i \in [BaseRec - ConfRec]$ do
        $td, revconf = Revconf[i]$;
        Remove the prediction of $td$ from $Temp\_Pred$;
    end
    return $Temp\_Pred$;
end
```

Note that in the measurement (line 3-6), α is the weight of the volume, and $(1-\alpha)$ is the weight of the frequency. The differential distance d_{td} can assess the sparsity around the query td and thus the level of distinctiveness of the query td. Then, we sort the queries in descending order by d_{td} (line 7). We regard the top *BaseRec* queries in d_{td} as the most distinctive queries and the attack target in the first module.

Finally, we recover top *BaseRec* queries (line 9-13). Given a query td_j, we calculate the distance $s(td_i, w_j)$ between the query td_i in real data and any keyword w_j in similar data. We define the $s(td_i, w_j)$ as

$$s(td_i, w_j) = \alpha \cdot |v_{td_i} - v_{w_j}| + (1-\alpha)|f_{td_i} - f_{w_j}|. \qquad (2)$$

We then recover the query td_i as

$$Pred(td_i) = \arg\min_{w_j \in W_s} s(td_i, w_j) \qquad (3)$$

By properly adjusting the weight of volume and frequency information, this module can recover the distinctive queries with high accuracy (e.g., averagely 77% in Enron when $BaseRec = 100$, see Section 5.1). We use the L1 norm here. It's worth noting that we also tested other norms and found that the L1 norm yields the best performance. We note that Zipf's law could not be applicable to certain datasets, such as those containing randomly generated texts or artificially padded datasets. In this case, the module's performance could be negatively affected. Nonetheless, it is uncommon for real-world datasets to deviate significantly from Zipf's law. Furthermore, padding a dataset to the extent that the volume of keywords diverges the Zipfian distribution would result in an inflated storage cost. Though, we can still achieve relatively high accuracy as demonstrated in Section 7.

4.3 Adjustment by Query Co-occurrence

In the second module (see Algorithm 2), we utilize the co-occurrence matrix to further refine the recovered queries output by Algorithm 1. Note that the attack accuracy of all queries relies on the precise recovery of the distinctive queries. Any incorrect recovery could significantly impact the overall accuracy.

Before launching the second module, we set the following input parameters:

- The recovered queries *Pred* and its cardinality *BaseRec* from the first module.
- The co-occurrence matrix C'_r of queries and C'_s of keywords in *Pred*. We construct the two co-occurrence matrices by first extracting the columns and rows in the co-occurrence matrix C_r and C_s when the corresponding queries and keywords appear in *Pred*. Then each row of C'_r and C'_s is normalized by dividing the sum of that row.
- The parameter *ConfRec* ($\leq$ *BaseRec*), which reflects the number of recovered queries after refinement. Similar to the *BaseRec*, a smaller value of *ConfRec* results in higher accuracy and a reduced number of recovered queries. In some restricted scenarios where we are only given a little prior knowledge or the leakage patterns have been "noised" by countermeasures, we should set the *ConfRec* to a smaller value to capture high accuracy.

With the above input, the module can verify the recovered queries in *Pred* and output the predictions of *ConfRec* queries with higher accuracy through the following process.

If most of the predictions in *Pred* are accurate, then for a correct prediction (td_i, w_i) in *Pred*, $C'_r[i]$ should be similar to $C'_s[i]$; otherwise, td_i and w_i are only similar in terms of volume and frequency and the relevant in co-occurrence matrix is not significant. Such similarity and deviation of $Pred[i]$ can be captured by calculating the Euclidean norm of $C'_r[i] - C'_s[i]$. We define *revconf* as the reversed confidence of a prediction (td_i, w_i) as

$$revconf = ||C'_r[i] - C'_s[i]||. \qquad (4)$$

If a prediction provides a smaller value of *revconf*, then it is considered more confident.

Based on the *revconf*, we calculate *Revconf* containing $(td, revconf)$ for all $td \in Td'$ (line 4-8). Then, we sort $Revconf$ in descent order (line 9). To provide $ConfRec$ verified predictions, we remove the top $BaseRec - ConfRec$ queries from $Pred$ and return the remaining predictions (line 10-14).

At the expense of recovering a smaller number of queries, Algorithm 2 can reach almost perfect accuracy. We show in Section 5.1 that when the $BaseRec$ is set to 100, and the $ConfRec$ is 20, we obtain 96.9% and 100% accuracy in Enron and Wikipedia, respectively.

4.4 Dynamic Recovery for All Queries

The prior modules provide predictions for a subset of queries. In Algorithm 3, we present the last module of our attack that leverages the relation between the recovered distinctive queries and the remaining queries. This module recovers queries through an iterative approach, where the recovered queries by the second module serve as known queries.

The module takes the following information as input and outputs the predictions for all queries.

- The predictions $Pred$ from the second module.
- The co-occurrence matrices C_r and the query list Td_r from the leakages.
- The co-occurrence matrices C_s and the keyword universe W_s from the similar data.
- The parameter $RefSpeed$, which controls the number of recovered queries in each iteration.

We denote the $unknownTd$ as currently un-recovered queries and the $unpairedW$ as currently unpaired keywords. We denote C_r^s and C_s^s as sub-matrices of C_r and C_s, which represent the co-occurrence matrix between un-recovered and recovered queries, and between unpaired and paired keywords, respectively. We normalize each row of C_r^s and C_s^s by dividing the sum of that row at each time the set of recovered queries changes.

We use the matrix C_r^s, C_s^s, and the distance s to evaluate the score between an un-recovered query td and an unpaired keyword w. If a row of C_s^s is similar to one of C_r^s, it might indicate a correct prediction for the corresponding keyword and query. The score contains two parts, the L2-norm of $C_r^s[td] - C_s^s[w]$ and the distance $s(td, w)$ (calculated in Equation 2), which are summed with weight β and $(1-\beta)$. The score of a prediction (td_i, w_j) is defined as:

$$score(td_i, w_j) = -\ln(\beta||C_r^s[td_i] - C_s^s[w_j]|| + (1-\beta)s(td_i, w_j)). \tag{5}$$

If a prediction (td, w) is correct, the $s(td, w)$ and the $||C_r^s[td] - C_s^s[w]||$ will be small, which results in a high score.

Algorithm 3: Dynamically recovering all queries.

```
1  procedure RecoverAll($Pred, Td_r, W_s, C_r, C_s, RefSpeed$)
2      $Final\_Pred \leftarrow Pred$;
3      $unknownTd \leftarrow Td_r - Final\_Pred.td$;
4      $unpairedW \leftarrow W_s - Final\_Pred.w$;
5      Extract $C_r^s$ and $C_s^s$ from $C_r$ and $C_s$, respectively; ▷ $C_r^s$ and $C_s^s$ is
         the co-occurrence matrix between $unknownTd$ and recovered
         queries, and between $unpairedW$ and paired keywords,
         respectively ;
6      while $unknownTd \neq \emptyset$ do
7          $Temp\_Pred \leftarrow \emptyset$;
8          for all $td \in unknownTd$ do
9              $Cand \leftarrow \emptyset$; ▷ $Cand$ stores candidate matches for $td$;
10             for all $w \in unpairedW$ do
11                 $score =$
                     $-\ln(\beta||C_r^s[td] - C_s^s[w]|| + (1-\beta)s(td, w))$;
12                 Add $(w, score)$ to $Cand$;
13             end
14             Sort $Cand$ in descending order according to the
                 $Cand.score$;
15             $certainty = Cand[0].score - Cand[1].score$;
16             Add $(td, Cand[0].w, certainty)$ to $Temp\_Pred$;
17         end
18         if $|unknownTd| < RefSpeed$ then
19             Add all predictions in $Temp\_Pred$ to $Final\_Pred$;
20         else
21             Add the $RefSpeed$ predictions in $Temp\_Pred$ with
                 largest $certainty$ to $Final\_Pred$;
22         end
23         Update $unknownTd$, $unpairedW$, $C_r^s$ and $C_s^s$;
24     end
25     return $Final\_Pred$;
26 end
```

Inspired by the RSA [13], we use a similar concept - *certainty* - to measure the level of assurance in the prediction for a query. Given a query td, the prediction (td, w_i) is considered certain if $score(td, w_i)$ is much higher than the score of any other predictions for td. The *certainty* of a prediction (td, w_i) is defined as:

$$certainty(td, w_i) = score(td, w_i) - \max_{j \neq i} score(td, w_j). \tag{6}$$

For example, if an un-recovered query has scores of 2, 3, and 7 with all three unpaired keywords, then the certainty of this query with unpaired keywords is -5, -4, and 4, respectively. In each iteration, we exclusively recover the queries with the highest certainty predictions.

This module runs through multiple iterations, each consisting of three main processes:

1. For all un-recovered queries $unknownTd$, calculate the *score* of all predictions between $unknownTd$ and $unpairedW$. Then, based on the *score*, calculate the highest *certainty* along with the prediction of each query, and add them to $Temp\_Pred$. (line 8-17)

2. If the number of un-recovered queries is less than $RefSpeed$, then add all the predictions in $Temp\_Pred$ to $Final\_Pred$, else add $RefSpeed$ predictions with largest *certainty* in $Temp\_Pred$ to $Final\_Pred$ (line 18-22).

3. Update the $unknownTd$ and $unpariedW$. Update and normalize the C_r^s and C_s^s accordingly. (line 23)

In the initial iterative process, the recovered queries have high accuracy due to their elevated level of *certainty*. As the process further operates, subsequently recovered queries are also recovered with high precision, primarily because of the augmented correlation between these queries and those that have already been recovered in previous iterations.

In this module, we use a similar method as the one introduced in RSA [13]. However, our approach provides several crucial differences. First, while the RSA algorithm exclusively utilizes the co-occurrence matrix to calculate the *score*, we incorporate both volume and frequency information in our *score* calculation. What's more, the performance of RSA is constrained to the "pre-set" known queries of high volume from the outset [13]. In Jigsaw, we use the second module to actively collect and recover the high-volume queries and further feed them into the third module. Their high volume provides Jigsaw with an advantage in query recovery. Moreover, for a query td, the RSA calculates the *score* of all keywords, potentially resulting in matching the query to a keyword that is already paired with another query. Our approach only calculates the *score* of unpaired keywords. Furthermore, we normalize the co-occurrence matrix in each iteration, which differs from the RSA algorithm. These differences collectively contribute to a more robust and accurate outcome for our algorithm, particularly when encountering defenses (for example, under the obfuscation in CLRZ [10] in Enron, our attack achieves $> 80\%$ accuracy while the RSA only captures $< 40\%$.). We provide a detailed analysis of the advantages of our approach in Section 6 and Section 7.

5 Evaluations

We evaluate our attack under various metrics in real-world datasets to show its effectiveness. We use Python 3.95 to simulate and run codes in Ubuntu 22.04.1 with 16 cores of an Intel(R) Xeon(R) Gold 5120 CPU (2.20GHz) and 64 GB RAM. Our code is publicly available in `https://github.com/JigsawAttack/JigsawAttack.git`.

5.1 Experimental Setup

Datasets. We utilize three datasets, Enron, Lucene, and Wikipedia, for our experiments. The Enron email corpus [44] was collected between 2000-2002, consisting of 30,109 emails, which is a widely used dataset in previous research. Lucene mailing list was formed between 2001-2020, with 66,491 emails from Apache Foundation [15]. For both Enron and Lucene datasets, we utilize pre-processed versions available in [31]. We use Wikipedia dataset [16] in 2020 and extract a subset of 1,000,000 documents by the algorithm in [37]. We employ the NLTK package [40] in Python to obtain all English words in datasets except the stop words for keyword extraction. In the experiments, we assume that the attacker obtains the same keyword universe as the user.

Frequency information. For the tests in Enron and Lucene datasets, we adopt the Google Trend [19], which contains 260 weeks of search trends in Google between October 2016 and October 2021, to generate query frequency for each keyword. Specifically, we calculate the sum of each query frequency in 1 to 50 weeks as the attacker's auxiliary knowledge. We normalize each keyword's frequency by dividing the frequency sum of all keywords as F_s. We also generate the user's queries according to the summed frequency in $1+\tau$ to $50+\tau$ weeks (denote as F) where τ is the time offset between the attacker's knowledge and the observation. For tests in the Wikipedia dataset, we use the Pageviews Analysis [27], which contains 75 months of page views from July 2015 to September 2021. We use the sum of each query frequency in 1 to 30 months as the attacker's auxiliary knowledge and the frequency of $1+\tau$ to $30+\tau$ months to generate queries.

Attacker's knowledge. We randomly divide all documents into two disjoint subsets of equal size. We use one subset as the user's encrypted database (i.e., the real data D_r) and another as the similar data D_s. Then, the user generates single-keyword queries according to the frequency F. The attacker generates W_s, V_s, and C_s from similar data and observes all the user's queries to obtain Td_r, V_r, F_r, and C_r. We perform 30 independent simulations. In each simulation, we randomly select half of the documents as similar data and generate queries according to F.

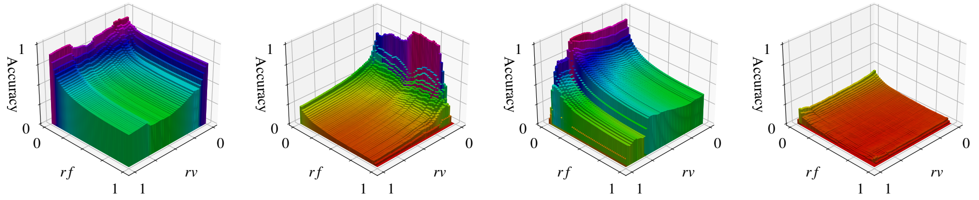
Accuracy definition. We use the terms *accuracy* and *recovery rate* to evaluate the attack performance. The recovery rate refers to the proportion of recovered queries in all observed queries (i.e., $|Recovered(Td^s)|/|Td^s|$). The accuracy denotes the correctly recovered queries out of recovered queries (i.e., $|CorrectRec(Td^s)|/|Recovered(Td^s)|$).

5.2 Performance of Algorithm 1 and 2

We here provide evaluations for Algorithm 1 and 2 (the results of the entire Jigsaw are in Section 5.3 and after). We first demonstrate the results of Algorithm 1 in four quadrants (including HVHF, HVLF, LVHF, and LVLF). For our experiments, we extract the top $1,000$ keywords based on their volume and generate $100,000$ queries with $\tau = 0$.

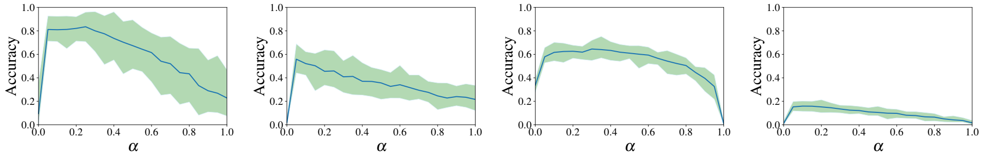
To evaluate the recovery in different quadrants, we sort the queries in Td_r according to their volume in descending order. We treat the first $rv \cdot l$ queries as high-volume queries ($l = |Td_r|$), while the remaining are low-volume queries. Similarly, we divide the queries into high and low-frequency queries, containing $rf \cdot l$ and $(1-rf) \cdot l$ queries, respectively. Using the above division, we categorize the queries into the HVHF, HVLF, LVHF, and LVLF quadrants. We set α to 0.5 and *BaseRec* to l to recover all queries and test the accuracy in four quadrants by varying rv and rf. Figure 2 demonstrates the accuracy of Algorithm 1 on Enron. Detailed results for Lucene and Wikipedia are given in the full version [29].

Figure 2(a) depicts the results of the HVHF quadrant in



(a) HVHF (b) HVLF (c) LVHF (d) LVLF

Figure 2: The accuracy of Algorithm 1 in four quadrants with different rv and rf, where we treat keywords with top-$rv \cdot l$ highest volume as high-volume keywords and treat keywords with top-$rf \cdot l$ highest frequency as high-frequency keywords (A larger rv means more queries are considered as high-volume queries. Similarly, a larger rf yields more queries that are categorized as high-frequency queries).



(a) HVHF (b) HVLF (c) LVHF (d) LVLF

Figure 3: The accuracy of Algorithm 1 on four quadrants with different α, where α is the weight of volume and $(1-\alpha)$ is the weight of frequency in measurement.

Table 2: Results of Algorithm 1 and Algorithm 2 on Enron, Lucene, and Wikipedia. The $ConfRec$ in Algorithm 2 equals to $BaseRec \times 100\%$, $BaseRec \times 50\%$, and $BaseRec \times 20\%$ respectively.

Dataset	$BaseRec$	$ConfRec/BaseRec \times 100\%$ (accuracy/recovery rate/correctly recovered number)[1]		
		100%[2]	50%	20%
Enron	25	94.14%/23.25%/23.0	100.00%/3.46%/12.0	100.00%/0.98%/5.0
	100	77.33%/43.14%/57.6	91.53%/17.18%/39.5	96.95%/5.48%/19.0
	400	51.85%/80.10%/112.2	72.06%/46.41%/81.1	83.96%/17.37%/50.2
Lucene	25	99.36%/27.51%/24.6	99.98%/4.17%/12.0	100.00%/1.93%/5.0
	100	86.31%/48.64%/76.3	99.58%/25.55%/48.9	99.80%/5.85%/19.9
	400	63.07%/83.97%/147.7	82.85%/50.04%/103.6	96.55%/27.64%/67.1
Wikipedia	25	99.71%/11.89%/23.8	100.00%/2.58%/12.0	100.00%/0.83%/5.0
	100	91.57%/28.67%/80.9	99.61%/15.04%/47.7	100.00%/3.59%/19.9
	400	68.79%/58.17%/177.3	88.87%/34.47%/120.2	97.54%/15.94%/64.3

[1] Each result is presented as (accuracy/recovery rate/correctly recovered number). The accuracy denotes the percentage of correctly recovered queries out of recovered queries. The recovery rate is the percentage of recovered queries out of all queries. The correctly recovered number is the number of correctly recovered and distinct queries.

[2] This column shows the results of Algorithm 1 as Algorithm 2 does not remove any predictions.

Enron. When $rv < 0.2$ or $rf < 0.1$, the accuracy is approximately 70%. As the increase of rf or rv indicates a greater proportion of queries with lower volume and frequency within the quadrant, the accuracy falls. This indirectly proves that queries with high frequency or high volume are easier recoverable. Similar trends can be observed in the performance of Lucene and Wikipedia. In contrast, the accuracy in the LVLF quadrant is only < 0.3 (see Figure 2(d)). From Figure 1, we can see that the queries in this quadrant are much denser as compared to other quadrants. There is a lack of distinguishability based on volume and frequency, resulting in such a low accuracy. Figure 2(b) exhibits the accuracy of queries confined to the HVLF quadrant, representing the top $rv \cdot l$ and bottom $(1-rf) \cdot l$ queries w.r.t. volume and frequency, respectively. The recovery of queries mainly relies on volume, and the accuracy reaches the summit when $rv < 0.1$ and $rf < 0.8$, which implies that queries with high volume and low frequency can be recovered with high accuracy. The LVHF quadrant delivers a similar result to the HVLF quadrant, see Figure 2(c). When rf is < 0.2, the recovery provides high accuracy, approaching 70%.

We also investigate the impact of the parameter α on the accuracy in the four quadrants. We set $rv = 0.1$, $rf = 0.1$, and $BaseRec = l$ to recover all queries, and the results are shown in Figure 3. The accuracy decreases when α is either 0 or 1, indicating that relying solely on frequency or volume for query recovery leads to poor accuracy. When α is appropriately configured, the accuracy $> 50\%$ in the HVHF, HVLF, and LVHF quadrants, while it remains relatively low in the LVLF quadrant. We also observe that the α displays a distinct impact on the accuracy in the quadrants. For example, in the HVLF quadrant, when $\alpha = 0.05$, we achieve the highest accuracy, while the LVHF quadrant's best performance is when α is about 0.3. In the HVLF quadrant, we achieve the highest accuracy when $\alpha = 0.05$, while in the LVHF quadrant, the best performance is obtained with α around 0.3. This suggests that selecting an appropriate value of α accordingly can lead to higher accuracy in different scenarios.

Table 2 presents the recovery results of Algorithm 1 and 2 when we consider $BaseRec \in \{25, 100, 400\}$ and $ConfRec/BaseRec \in \{100\%, 50\%, 20\%\}$. Overall, the accuracy achieved for various parameter combinations surpasses 50%. As $BaseRec$ and $ConfRec/BaseRec$ decrease, indicating a reduction in the number of recovered queries, the recovery rate decreases while the accuracy exhibits an increase. For instance, when we set $BaseRec = 25$ and $ConfRec/BaseRe = 20\%$, Algorithm 2 achieves around 1% recovery rate but 100% accuracy on all the datasets. We also observe that with the same parameters, the results in Wikipedia are better than those in Enron and Lucene, indicating that Wikipedia contains more distinctive queries.

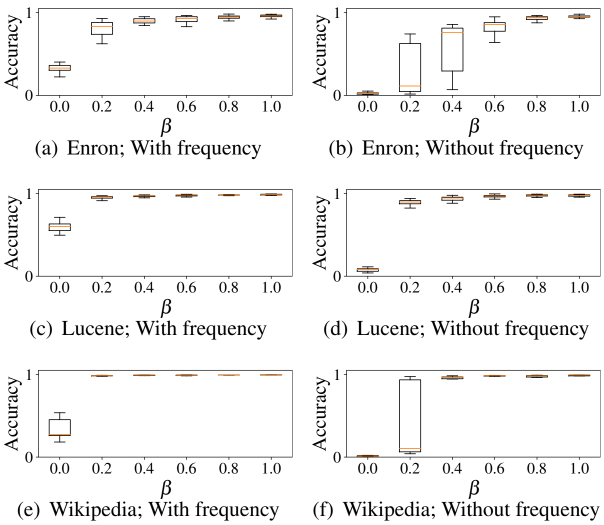
Despite a decline in the recovery rate, when Algorithm 2 is employed, its accuracy remains remarkably high with a small $BaseRec$ and $ConfRec$. In comparison to RSA [13] relying on a dozen known queries and their relations (with other queries) to recover all queries accurately, the refinement on queries proposed by Algorithm 2 is sufficient to pose a significant threat to all the user's queries. We demonstrate this in detail in the experiments of the next subsection.

5.3 Results of the Jigsaw Attack

We show the results of the Jigsaw attack and demonstrate how β influences the accuracy here. We use the same setting as in Section 5.1. Besides, we set $BaseRec$ to 100, $ConfRec$ to 50, α to 0.3, and $RefSpeed$ to 15.

The experimental results when varying the values of β in Algorithm 3 are depicted in the first column of Figure 4. When $\beta = 0$, meaning the recovery only relies on the frequency and volume, the accuracy reaches around 30% in Enron and Wikipedia and 60% in Lucene. As β increases, we see the rise in accuracy, ultimately reaching the peak of over 95% accuracy for Enron and over 98% for Lucene and Wikipedia.

As demonstrated above, using similar documents and knowledge of query frequency, Jigsaw achieves $> 95\%$ accuracy. However, there are some cases where the attacker may not have access to query frequency, such as when dealing with newly established databases. To examine the effectiveness of Jigsaw under such circumstances, we perform additional evaluations without utilizing frequency information. In the absence of frequency information, the attacker should discern fewer distinctive queries, yielding a negative influence on the performance of Algorithm 1 and Algorithm 2. However, by adjusting the parameter $ConfRec$ to recover a smaller number of queries, the first two modules can still achieve high accuracy in identifying distinctive queries. We note that with this smaller yet highly accurate set of recovered queries, our attack's accuracy still remains stable and reaches $> 90\%$ when β is set to 0.8 and 1.0 (See in the second column of Figure 4).



(a) Enron; With frequency (b) Enron; Without frequency

(c) Lucene; With frequency (d) Lucene; Without frequency

(e) Wikipedia; With frequency (f) Wikipedia; Without frequency

Figure 4: The accuracy of Jigsaw with different β, where *beta* is the weight of co-occurrence information and $(1-\beta)$ is the weight of volume and frequency information in calculating *score*. The left and right columns display the results with and without frequency information.

5.4 Durability

Table 3: Results of recovery accuracy with outdated frequency in different τ, where τ is the time offsets between attacker's prior knowledge of the frequency and user's queries (measured in weeks for Enron and Lucene, and in months for Wikipedia).

Dataset	$\tau = 10$w	$\tau = 50$w	$\tau = 100$w	$\tau = 150$w
Enron	0.9279	0.9150	0.8881	0.8824
Lucene	0.9959	0.9963	0.9955	0.9897
Dataset	$\tau = 2$m	$\tau = 10$m	$\tau = 20$m	$\tau = 30$m
Wikipedia	0.9959	0.9919	0.9721	0.9608

In the experiment, we use an "outdated" query frequency obtained from the past as auxiliary information to enhance accuracy. We here introduce the concept of *durability* to measure the effect of the time offset between the outdated and target queries on the attack's recovery. The time offset indicates how "old" the query frequency information is. An outdated piece of frequency information might deviate significantly from the actual query frequency, possibly leading to the failure of attacks. We consider an attack to be durable if it can maintain its accuracy even as the time offset increases.

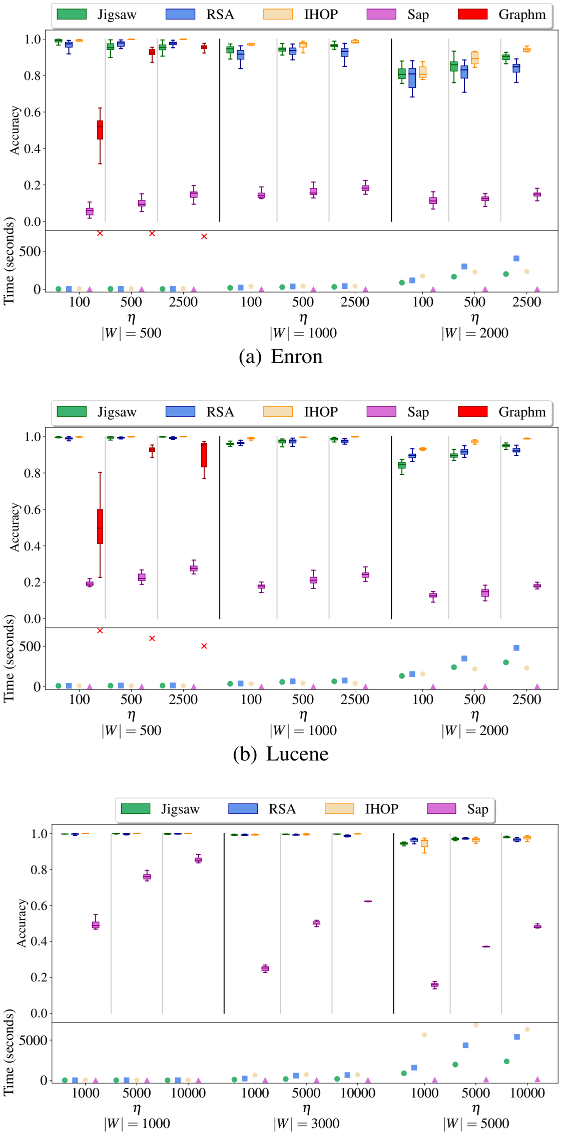
We conduct experiments to evaluate the durability of our attacks in Table 3. For Enron and Lucene, we use the frequency of the first 50 weeks in Google Trend as the attacker's auxiliary information, while the target queries are generated using the frequency during τ and $50+\tau$ weeks. For Wikipedia, we use the first 30 months' query frequency in Pageviews Analysis as the auxiliary information, and the queries are generated according to the frequency during τ and $30+\tau$ months. Note that τ is the corresponding time offset. In the time offset between 10 and 150 weeks, the drop of the attack accuracy is rough > 0.05 in Enron and Lucene. In Wikipedia, the accuracy decreases about 3.5% as the time offset increases from 2 to 30 months. These results suggest that a leaked query frequency continues to have an impact on our attack even after several years have passed.

5.5 Summary of Evaluations

The results clearly illustrate that the first module of Jigsaw successfully recovers queries with a high level of accuracy. We also confirm that the second module can obtain nearly 100% accuracy for dozens of queries, and the last module is able to recover all queries with about 95% accuracy. Even without the frequency information, the accuracy does not experience a significant decline. At last, Jigsaw demonstrates its durability by maintaining consistent accuracy, even as the time offset increases from several weeks to years.

6 Comparisons with Other Attacks

We compare the performance of our attack with Graph Match attack [35] (Graphm), Sap attack [31] (Sap), Refine Score attack [13] (RSA) and IHOP attack [32] (IHOP). We do not test the Freq attack [26] here as the Sap dominates its results.



(a) Enron

(b) Lucene

(c) Wikipedia

Figure 5: Accuracy & Time comparisons in Enron, Lucene, and Wikipedia.

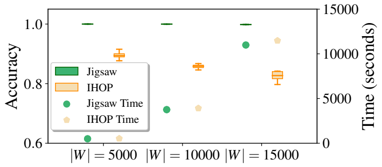


Figure 6: Accuracy comparisons with IHOP within similar runtime.

6.1 Settings and Parameters

We extract the top $|W|$ keywords based on their volume as keyword universe. In Enron and Lucene, we evaluate the above attacks for different values of $|W|$, namely 500, 1000, and 2000. But for Graphm, we omit $|W| = 1000, 2000$ due to the extended time required for the test. For example, the running time for Graphm with $|W| = 1000$ exceeds 10,000 seconds. We suppose the attacker knows the frequency of each keyword in the keyword universe in the first 50 weeks and generate η queries each week for a duration spanning from 50 to 100 weeks ($\tau = 50$) using the frequency obtained from Google trend. We set η to 100, 500, and 2500. In Wikipedia, we test all the attacks except the Graphm, and we set the $|W|$ to 1000, 3000, and 5000. We assume the attacker knows the initial 30 months' query frequency from Pageviews Analysis, and the user generates η queries based on the frequency during the period of 10 and 40 months ($\tau = 10$). We set the η to 1000, 5000, and 10000.

Parameters for Jigsaw. Recall that the selections of α, β, $BaseRec$, and $ConfRec$ can significantly influence Jigsaw's accuracy (see Section 5). We briefly introduce the reasons behind these parameters selection.

- As for α, we use it to control the weight of volume and frequency information. Based on Figure 3, it is recommended to select the parameter from 0.05 to 0.4. If the volume information is not accurate (i.e. being noised by certain countermeasures), a smaller α is recommended, and vice versa. Here, we set the α to 0.3.
- For β, one may choose β from 0.8 to 1.0 as illustrated in Figure 4. Note that as a larger β indicates assigning more weight to co-occurrence rather than volume and frequency, the co-occurrence information appears to play an important role in recovering queries. Similar to the case of α, if the co-occurrence is affected by noise, one can opt for a relatively small β. We set the β to 0.9 here.
- For $BaseRec$ and $ConfRec$, we found that the $ConfRec$ should be set to at least 5 for the third module to initiate, and the $BaseRec$ should range from $1.2 \times ConfRec$ to $2 \times ConfRec$. In normal SSE settings, $BaseRec$ and $ConfRec$ should be sufficiently large to output more accurately recovered queries, whereas they are set to small when observed information is noised to ensure the accuracy of the second module. The $BaseRec$ and $ConfRec$ are set to 45 and 35, respectively.
- When it comes to $Refspeed$, it controls how many queries to recover in each iteration. One may use a large $RefSpeed$ to optimize the runtime of Jigsaw (such as one-tenth of $|W|$), but this could harm attack accuracy. A gradually increased $RefSpeed$ is recommended when dealing with countermeasures, as it can yield both practical runtime and accuracy. The $RefSpeed$ is set to 10 when the keyword universe is small ($<= 2,000$) and 50 when the universe is large ($> 2,000$). As in the time-limited settings, we set the $Refspeed$ to $|W|/10$.

Parameters for other attacks. We use the implementation of PATH algorithm [49] available in the package[1] to solve the graph matching problem in Graphm (aiming to produce the best performance). We set the α in Graphm to 0 because we find that Graphm can perform its best when $\alpha = 0$ in our settings. Note we conduct tests ranging from $\alpha = 0$ to $\alpha = 1$, with increments of 0.1. Recall that RSA requires some known queries in the setup. We randomly choose 10 queries and reveal the true keywords for RSA. We also include the results of RSA with varying numbers of known queries in the full version [29]. The refine speed in RSA is set to the same as Jigsaw. We set the α in Sap to 0.5. For IHOP, we set the p_{free} to 0.25 and the n_{iters} to 500.

6.2 Comparison Results

Comparisons with all tested attacks. Here, we demonstrate the results of all tested attacks in Figure 5. Our attack provides comparable accuracy to RSA and IHOP while showing a significant advantage over Sap and Graphm. Graphm's accuracy is low upon $\eta = 100$ but improves if more queries are observed. It is argued that Graphm requires observation of almost all possible queries to achieve high accuracy [31], and it also takes longer matching time between queries and keywords. On the other hand, Sap solely utilizes the frequency and volume information, resulting in relatively lower accuracy that increases as it observes more queries. We also observe that increasing $|W|$ results in a decrease in all attacks' accuracy. This is because a larger value of $|W|$ introduces a greater number of low-volume keywords.

In the context where a large number of queries are observed, our attack can outperform RSA. As the increase of this number, our attack delivers a boost in accuracy. But RSA cannot gain advantages from more queries because it does not leverage the frequency information. Our performance is similar to that of IHOP, as both of the attacks exploit the frequency and co-occurrence of queries. We observe that IHOP achieves slightly higher accuracy than our attack. This can be attributed to its random fixing and free strategy, which enhances the matching of a portion of the queries in each iteration. This strategy consumes a significant amount of time, especially when the keyword universe is large. For example, when $|W|$ is 5000 in Wikipedia, IHOP takes approximately twice as long as Jigsaw to complete the attack. We are going to show in Section 7 that the strategy used by IHOP is not robust once certain countermeasures introduce noise to the leaked information.

Comparisons with IHOP in large keyword universe under the same time limitation. We also present the performance of Jigsaw and IHOP within the same time limitation when the keyword universe is large. Before the evaluations, we adjust the parameters by setting $RefSpeed$ to $|W|/10$ for Jigsaw and restricting n_{iters} for IHOP to keep both runtimes at a similar pace. Note that without any adjusting, IHOP could take approx. 24,000 seconds for 100 iterations, and nearly five times that for 500 iterations, when $|W| = 10,000$. We illustrate the results under Wikipedia in Figure 6. As the keyword universe increases, from 5,000 to 15,000, the runtime costs of both attacks jump from $< 1,000$ seconds to nearly 10,000 seconds. IHOP demonstrates a continuous fall in accuracy, from roughly 89% to 85%, while Jigsaw's accuracy stands at the same level, $10-15\%$ higher than that of IHOP.

[1] http://projects.cbio.mines-paristech.fr/graphm/

7 Against Countermeasures

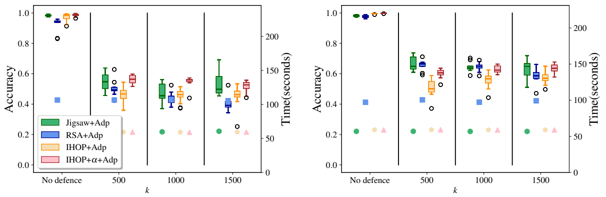
We evaluate the attacks against the padding in CGPR [5] and obfuscation [10]. We also show the results against the padding in SEAL [14] and the cluster-based padding [3,43] in Appendix C. We specifically compare the attacks against the padding strategy employed in SEAL, rather than the "entire" SEAL (i.e., padding + ORAM)[2] .

To counter obfuscation, Oya et al. [32] proposed an adaptation for IHOP, which modifies the co-occurrence matrix of keywords in similar data. We apply the same philosophy to Jigsaw and RSA. Note that we also design adaptations on the compared attacks against the padding. These adaptations can effectively minimize the difference between the user's data after the noise injection and the similar data. We highlight that the accuracy of RSA and IHOP increases significantly in most situations after applying the adaptations (such as 30% improvement against the padding in CGPR). We present a comprehensive overview of the adaptations and the performance of RSA and IHOP with/without the adaptations in Appendix B. We emphasize that there have been no systematic and proper studies for optimal adaptions in existing attacks, rendering this as an interesting open problem.

We conduct a comparative analysis of our attack, RSA [13], and IHOP [32] w.r.t. the aforementioned countermeasures, in Enron, Lucene, and Wikipedia. We do not test the SAP and Graphm in this section, as they exhibit relatively poor performance or require excessive computational time for the attacks. We also introduce an α term into the objective function of IHOP to balance the weight between the frequency and volume terms against countermeasures (noted as IHOP-α). We test the α from 0 to 1 with a step length of 0.1, and in most cases, $\alpha = 0.1$ brings the best results for IHOP-α. We fix $\alpha = 0.1$ for IHOP-α in the following presentations.

For Enron and Lucene, we set $|W|$ to 1000 and set η to 500. The attacker is allowed to observe η queries per week over a duration of 50 weeks. In Wikipedia, we set the $|W|$ to 1000, 3000 and 5000 and η to 5000, and the attacker can observe η queries per month for a total duration of 30 months. The time offset τ is set to 0. For Jigsaw, to account for the injection of noise to the volume (by the countermeasures), we set α to a relatively small value, 0.2. And we set β to 0.9. The *BaseRec* and *ConfRec* are also set to relatively small, 15 and 10, to ensure that the second module of Jigsaw can produce correct recoveries. We set the *RefSpeed* of our attack and RSA to 5 in Enron and Lucene. In Wikipedia, we use a gradually increased value to shorten the runtimes of Jigsaw and RSA. For each iteration of Jigsaw's third module and RSA, the value increases by 10%. The known query number is set to 15 in RSA. The n_{iters} and p_{free} are set to 500 and 0.25 in IHOP and IHOP-α.

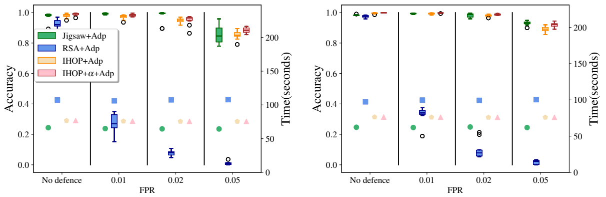
7.1 Against the Padding in CGPR



(a) Enron

(b) Lucene

Figure 7: Comparisons with RSA and IHOP against the padding in CGPR [5], in Enron and Lucene.



(a) Enron

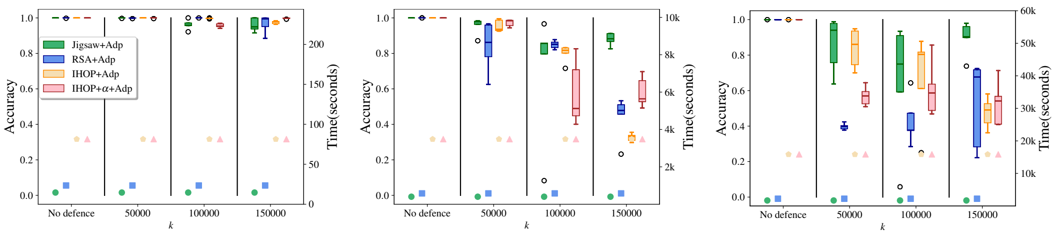
(b) Lucene

Figure 8: Comparisons with RSA and IHOP against the obfuscation in CLRZ [10], in Enron and Lucene.

Padding in CGPR [5] injects fake documents to increase the query volume to the nearest multiple of k. This strategy adds noises to the volume and access pattern and at the same time increases the communication and storage costs (see the full version [29] for experimental results). For queries with low volume, the padding can substantially change the access and volume patterns, and the volume is more likely to be "expanded" to the same "length". But the high-volume queries, on the other hand, are poorly protected. This is so because, for those queries (already with a "large" volume), the (on-top) padding volume could be relatively small. In this case, the impact on the leakage patterns of high-volume queries is minimal. We state that since the first two modules of Jigsaw concentrate on recovering high-volume and high-frequency queries that are not significantly affected by the padding, they can keep producing highly accurate predictions. By leveraging these accurate recoveries, we can gain "more" pre-knowledge to pose a severe threat to low-volume queries. In Figure 7, we have $k \in \{500, 1000, 1500\}$ for Enron and Lucene; while in Figure 9, k is set much larger, $\in \{50000, 100000, 150000\}$. This is because Wikipedia contains a significantly higher number of documents than other datasets.

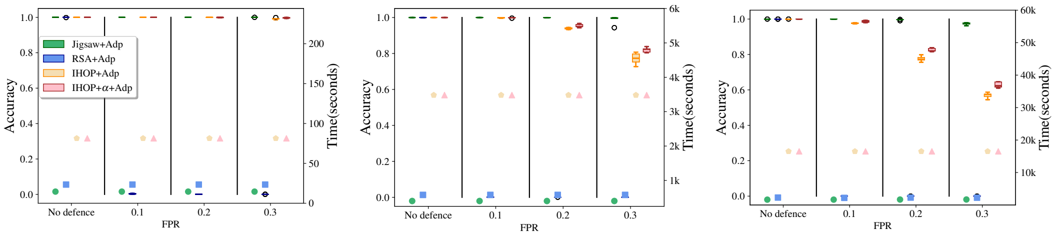
All the tested attacks exhibit similar accuracy, about 50% in Enron and $> 60\%$ in Lucene. In Wikipedia, the gap in accuracy is more noticeable, and Jigsaw demonstrates a significant advantage over others when $|W| \geq 3000$. For example, when $k = 150,000$ and $|W| = 3000$, Jigsaw provides nearly 90%

[2] We note that ORAM, another crucial component within SEAL, can be used to hide the access and search pattern, and thus could probably counter all the attacks in Table 1.



(a) $|W| = 1000$ (b) $|W| = 3000$ (c) $|W| = 5000$

Figure 9: Comparisons with RSA and IHOP against the padding in CGPR [5], in Wikipedia.



(a) $|W| = 1000$ (b) $|W| = 3000$ (c) $|W| = 5000$

Figure 10: Comparisons with RSA and IHOP against the obfuscation in CLRZ [10], in Wikipedia.

accuracy, whereas RSA and IHOP only obtain $< 60\%$. IHOP-α performs slightly better than IHOP, with a 60% accuracy. This above result is attributed to the fact that the padding cannot protect the distinctive queries in Wikipedia, allowing Jigsaw to recognize and further recover them, which gives it an advantage in recovery. We clearly observe that Jigsaw consumes significantly less runtime than IHOP and IHOP-α in Wikipedia due to a gradually increased $RefSpeed$.

7.2 Against the Obfuscation in CLRZ

We showcase the experimental results of attacks against the obfuscation in CLRZ [10], which works by indexing a keyword to documents that do not contain the keyword with probability FPR and removing the index of documents that do contain the keyword with probability TPR. Since the obfuscation does not involve padding, it does not affect storage costs. However, the communication costs will increase greatly due to a larger number of unrelated documents being retrieved (see the full version [29] for experimental results). In Figure 8 and 10, we set $TPR = 0.999$ and $FPR \in \{0.01, 0.02, 0.05\}$ in Enron and Lucene and $FPR \in \{0.1, 0.2, 0.3\}$ in Wikipedia.

Under the obfuscation, the accuracy of RSA drops abruptly to below 20%. Jigsaw just experiences a minor decrease as the FPR increases and, in most cases, maintains accuracy $> 85\%$ in all tested datasets. IHOP and IHOP-α perform similarly to Jigsaw in Enron and Lucene. But in Wikipedia, their accuracy drops significantly with a large $|W|$. When $FPR = 0.3$ and $|W| = 3000$, the accuracy only reaches about 80%, dropping to 60% when $|W| = 5000$. In contrast, Jigsaw remains an accuracy above 95% under the same settings.

7.3 Discussion

Under all tested countermeasures, it is evident that Jigsaw (after the adaptations) achieves the highest accuracy, $> 70\%$, in most cases. On average, RSA and IHOP could closely follow Jigsaw's performance. But they have some pitfalls. RSA is vulnerable to the obfuscation in CLRZ, resulting in $< 20\%$ accuracy. IHOP and IHOP-α also experience low accuracy against the countermeasures with a large $|W|$. Their accuracy is approx. 50% against the padding in CGPR (with $k = 150,000$) and about 60% against the obfuscation in CLRZ (with $TPR = 0.3$) on Wikipedia with $|W| = 5,000$. While one may have the option to apply the defenses to mitigate RSA and IHOP, Jigsaw proves to be a more "robust" attack that remains effective.

There are several countermeasures that might defend against Jigsaw. ORAM [45], a popular solution to SSE attacks, conceals the access pattern and the derived co-occurrence matrix, reducing the effectiveness of Jigsaw's second and third modules. But this comes with an $\Omega(\log N)$ amortized blowup of communication cost for databases of size N. Providing similar efficacy on the access pattern, PIR [21] is another potential option. However, it requires heavy server-side computation and does not support private updates by the client. Apart from completely hiding the access pattern, strong padding techniques may infuse noise into the volume pattern of the high-volume queries, making Jigsaw's first module unable to produce sufficient correct recoveries - resulting in low accuracy. A drawback of this solution is the necessity to pad a considerable amount of files, especially for high-volume queries. It remains an intriguing challenge to develop a padding technique that is both efficient and secure.

8 Conclusion

We propose the Jigsaw, a new similar-data attack against SSE, which works by first recovering the most distinctive queries and utilizing them to recover all queries further. We test Jigsaw in different datasets and showcase the stable accuracy of around 95% in query recovery. Moreover, our attack can provide an accuracy of about 60% and 85% against padding [5] and obfuscation [10], respectively, outperforming existing works [13, 26, 31, 32]. The proposed attack exposes the vulnerabilities of existing SSE schemes. Developing secure and practical SSE schemes that are resistant to such attacks is an open problem.

Acknowledgments

We would like to thank the shepherd and the anonymous reviewers for their valuable comments. This work was partly supported by the National Key Research and Development Program of China under Grant No. 2021YFB3101304, the National Natural Science Foundation of China under Grant No. 62372201 and No. 62272186, the Innovation Project of Jinyinhu Laboratory under Grant No. 2023JYH010103, and the European Union's Horizon Europe Research and Innovation Programme under Grant No. 101073920 (TENSOR), No. 101070052 (TANGO), and No. 101070627 (REWIRE).

References

[1] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *NDSS*, 2020.

[2] Raphael Bost. Σοφος: Forward secure searchable encryption. In *CCS*, 2016.

[3] Raphael Bost and Pierre-Alain Fouque. Thwarting leakage abuse attacks against searchable encryption - A formal approach and applications to database padding. *Cryptology ePrint Archive*, 2017. http://eprint.iacr.org/2017/1060.

[4] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *CCS*, 2017.

[5] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *CCS*, 2015.

[6] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO*, 2013.

[7] David Cash and Stefano Tessaro. The locality of searchable symmetric encryption. In *EUROCRYPT*, 2014.

[8] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In *CCS*, 2018.

[9] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, 2005.

[10] Guoxing Chen, Ten-Hwang Lai, Michael K. Reiter, and Yinqian Zhang. Differentially private access patterns for searchable symmetric encryption. In *INFOCOM*, 2018.

[11] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, 1995.

[12] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS*, 2006.

[13] Marc Damie, Florian Hahn, and Andreas Peter. A highly accurate Query-Recovery attack against searchable encryption using Non-Indexed documents. In *USENIX Security*, 2021.

[14] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. SEAL: attack mitigation for encrypted databases via adjustable leakage. In *USENIX Security*, 2020.

[15] Apache Foundation. Mail archieves of lucene, 1999. https://mail-archives.apache.org/mod_mbox/#lucene.

[16] Wikipedia Foundation. Wikipedia databases, 2020. https://www.wikipedia.org.

[17] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *CRYPTO*, 2016.

[18] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 1996.

[19] Google. Google trends, 2004. https://trends.google.com/trends/.

[20] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. Pancake: Frequency smoothing for encrypted data stores. In *USENIX Security*, 2020.

[21] Alexandra Henzinger, Matthew M Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *USENIX Security*, 2023.

[22] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *NDSS*, 2012.

[23] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *EUROCRYPT*, 2017.

[24] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *CCS*, 2012.

[25] Steven Lambregts, Huanhuan Chen, Jianting Ning, and Kaitai Liang. VAL: volume and access pattern leakage-abuse attack with leaked documents. In *ESORICS*, 2022.

[26] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-an Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 2014.

[27] Marcel Ruiz Forns MusikAnimal, Kaldari. Pageviews toolforge, 2015. https://pageviews.toolforge.org/.

[28] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. Dynamic searchable encryption via blind storage. In *S&P*, 2014.

[29] Hao Nie, Wei Wang, Peng XU, Xianglong Zhang, Laurence T. Yang, and Kaitai Liang. Query recovery from easy to hard: Jigsaw attack against SSE. *ArXiv*, 2024. https://arxiv.org/abs/2403.01155.

[30] Jianting Ning, Xinyi Huang, Geong Sen Poh, Jiaming Yuan, Yingjiu Li, Jian Weng, and Robert H Deng. Leap: Leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset. In *CCS*, 2021.

[31] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *USENIX Security*, 2021.

[32] Simon Oya and Florian Kerschbaum. IHOP: Improved statistical query recovery against searchable symmetric encryption through quadratic optimization. In *USENIX Security*, 2022.

[33] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Symmetric searchable encryption with sharing and unsharing. In *ESORICS*, 2018.

[34] Rishabh Poddar, Stephanie Wang, Jianan Lu, and Raluca Ada Popa. Practical volume-based attacks on encrypted databases. In *EuroS&P*, 2020.

[35] David Pouliot and Charles V Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *CCS*, 2016.

[36] Zhiwei Shang, Simon Oya, Andreas Peter, and Florian Kerschbaum. Obfuscated access and search patterns in searchable encryption. In *NDSS*, 2021.

[37] David Shapiro. Convert wikipedia database dumps into plaintext files, 2021. https://github.com/daveshap/PlainTextWikipedia.

[38] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *S&P*, 2000.

[39] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS*, 2014.

[40] Liling Tan Steven Bird. Nltk corpus, 2021. https://www.nltk.org/howto/corpus.html.

[41] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *CCS*, 2018.

[42] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE transactions on pattern analysis and machine intelligence*, 1988.

[43] Viet Vo, Xingliang Yuan, Shifeng Sun, Joseph K Liu, Surya Nepal, and Cong Wang. Shielddb: An encrypted document database with padding countermeasures. *TKDE*, 2021.

[44] CMU William W. Cohen, MLD. Enron email datasets, 2015. https://www.cs.cmu.edu/~./enron/.

[45] Zhiqiang Wu and Rui Li. OBI: a multi-path oblivious RAM for forward-and-backward-secure searchable encryption. In *NDSS*, 2023.

[46] Lei Xu, Huayi Duan, Anxin Zhou, Xingliang Yuan, and Cong Wang. Interpreting and mitigating leakage-abuse attacks in searchable symmetric encryption. *TIFS*, 2021.

[47] Lei Xu, Xingliang Yuan, Cong Wang, Qian Wang, and Chungen Xu. Hardening database padding for searchable encryption. In *INFOCOM*, 2019.

[48] Peng Xu, Willy Susilo, Wei Wang, Tianyang Chen, Qianhong Wu, Kaitai Liang, and Hai Jin. Rose: Robust searchable encryption with forward and backward security. *TIFS*, 2022.

[49] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008.

[50] Xianglong Zhang, Wei Wang, Peng Xu, Laurence T. Yang, and Kaitai Liang. High recovery with fewer injections: Practical binary volumetric injection attacks against dynamic searchable encryption. In *USENIX Security*, 2023.

[51] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security*, 2016.

[52] George Kingsley Zipf. *Human behavior and the principle of least effort: An introduction to human ecology*. Ravenio Books, 2016.

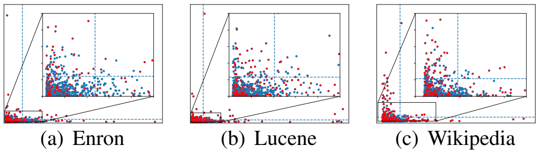

(a) Enron (b) Lucene (c) Wikipedia

Figure 11: The distribution of queries in normalized volume and frequency. The figures own the same format as Figure 1.

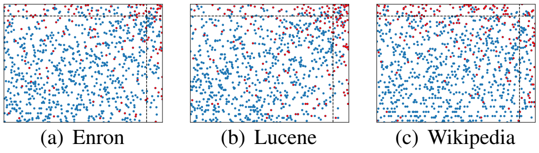
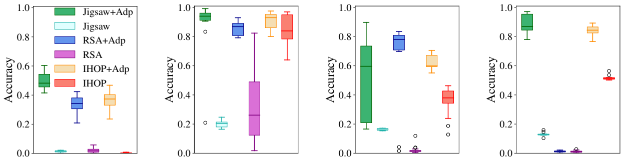

(a) Enron (b) Lucene (c) Wikipedia

Figure 12: The distribution of queries (showing their ranks based on volume and frequency). The dot and dashed lines share the same meaning as in Figure 1. Those at further to the right indicate higher rankings in volume; those moving upwards represent elevated ranking in frequency.

A Query Distribution and Simple Attack

We present a simple attack and its evaluation to showcase the relationship between query distribution and query recovery. The attack employs knowledge of frequency and volume information to pair queries with the keywords having the most similar frequency and volume. We assume the attacker knows a similar dataset D_s and generates a keyword universe $W_s = [w_1, w_2, \ldots, w_m]$. Then, it generates the corresponding volume $V_s = [v_{w_1}, v_{w_2}, \ldots, v_{w_m}]$ of each keyword from D_s. It also knows a historical query frequency $F_s = [f_{w_1}, f_{w_2}, \ldots, f_{w_m}]$ of W_s. Also, the attacker can observe the volume and search pattern of queries the user issues and generate the frequency F_r and volume V_r of them. After normalizing the F_s, V_s, F_r, and V_r, it pairs each query td_i with keyword w_j which has the smallest value of $|v_{w_j} - v_{td_i}| + |f_{w_j} - f_{td_i}|$.

We test the attack in Enron, Lucene, and Wikipedia. The results are shown in Figure 11 and 12. In Figure 11, we normalize the volume and frequency of each query. The horizontal dashed line divides the top 10% queries on volume from other queries, and the vertical dashed line divides the top 10% queries on frequency from other queries. The two lines divide the queries into four quadrants, i.e., the HVHF, HVLF, LVHF, and LVLF. The blue dots denote the queries, and the red dots denote the queries successfully recovered by the simple attack. In all tested datasets, the queries in the HVHF quadrant are sparse, and the attack has a high accuracy there. On the other hand, in the LVLF quadrant, the queries are nearly indistinguishable and hard to be recovered. We can zoom out the left low corner to show this more clearly. We rank the queries according to their volume and frequency and show the queries according to their rank in Figure 12. As the



(a) CGPR (b) Cluster-based (c) SEAL (d) Obfuscation

Figure 13: The results of RSA and IHOP with (+Adp) / without the adaptations on similar data against the padding in CGPR [5], the obfuscation [10], the cluster-based padding [3, 43], and the padding in SEAL [14].

queries have a higher rank in volume or frequency, the red dot is denser, showing a higher accuracy in recovery.

In Section 4, we provide the definition of *differential distance* d_{td} of a query. The distinctiveness of a query increases as its differential distance becomes larger. Based on d_{td}, we define the number K of distinctive queries in a dataset as:

$$K = |\{i : \frac{d_{td_i}}{\sum_{td_j \in W} d_{td_j} / |W|} > \lambda\}|. \tag{7}$$

In Enron, Lucene, and Wikipedia, setting $\lambda = 5$, we have that the number is roughly 18, 20, and 33, which can concur with the results in Section 5, 6, and 7.

B Adaptations to Similar Data

It seems that the countermeasures such as padding and obfuscation do not consider the protection of the parameters. If gaining access to the parameters, the attacker will be able to make adaptations to similar data to weaken the countermeasures. For example, in the case of padding in CGPR, the attacker can utilize the parameter k to pad the similar data, thereby minimizing the disparity between the similar data and the padded data. We say that this effectively mitigates the adverse effects of padding on query recovery. Specifically, our adaptations applied to Jigsaw, RSA, and IHOP are as follows.

- *Padding in CGPR* [5]. This strategy pads the query volume to the nearest multiple of k. We here employ the same padding approach with a different parameter, k_{sim}, on similar data. We calculate k_{sim} as k multiplied by the ratio of the sizes of the similar dataset (D_s) and the original dataset (D), i.e., $k_{sim} = k \cdot |D_s|/|D|$. Accordingly, this adjustment modifies ID_s, which subsequently affects the C_s and V_s in the attacks. We also apply the same strategy for RSA and IHOP.
- *Padding in SEAL* [14]. In SEAL, the volume distribution of the padded queries is closely related to the size of the dataset. The varying sizes of similar data to the user's data result in different volume distributions after padding. To adapt Jigsaw against the SEAL's padding, we generate a new similar data D'_s with the size $|D|$ (aligning with the size of the user's data D) by expanding D_s with its own data "copies" to keep the volume distribution of D_s.
- *Cluster-based padding* [3, 43]. We generate a new similar dataset D'_s by padding D_s with the same parameter and replace D_s with D'_s in attacks.
- *Obfuscation* [10]. We have two phases for this adaptation. Firstly, we apply the co-occurrence matrix (in Equation 8) in [32] to adapt the influence of obfuscation in similar data.

$$C_s^{obf}[i,j] = \begin{cases} TPR^2 \cdot C_s[i,j] + FPR^2 \cdot C_s^{not}[i,j] + \\ TPR \cdot FPR \cdot (1 - C_s[i,j] - C_s^{not}[i,j]),\ i \neq j; \\ TPR \cdot C_s[i,j] + FPR \cdot C_s^{not}[i,j],\ i = j. \end{cases} \tag{8}$$

where $C_s^{not} = (1 - ID_s)(1 - ID_s)^{\top}/|D_s|$. Then, we revise V_s as $V_s^{obf}[i] = TPR \cdot V_s[i] + FPR \cdot (1 - V_s[i])$ for further adaptation.

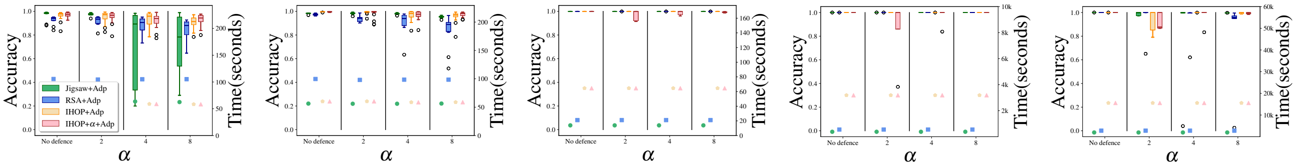
We then adopt the same padding strategy on D'_s and replace the D_s with the D'_s in attacks.

We test Jigsaw, RSA [13], and IHOP [32] with/without the adaptations on Enron against the CGPR's padding [5] ($k = 1500$), the obfuscation [10] (TPR$= 0.999$, FPR$= 0.05$), the cluster-based padding [43] ($\alpha = 8$), and the SEAL's padding [14] ($x = 4$). The parameters are the same as in Section 7. The results are presented in Figure 13. For obfuscation, RSA with/without the adaption performs poorly ($< 10\%$ accuracy). On the other hand, under three padding strategies, the proposed adaptations optimize the accuracy of RSA and IHOP significantly. Noticeably our Jigsaw attack with adaptations still takes the lead in most cases, see Section 7 for comparison details.

C Results Against Other Padding Strategies

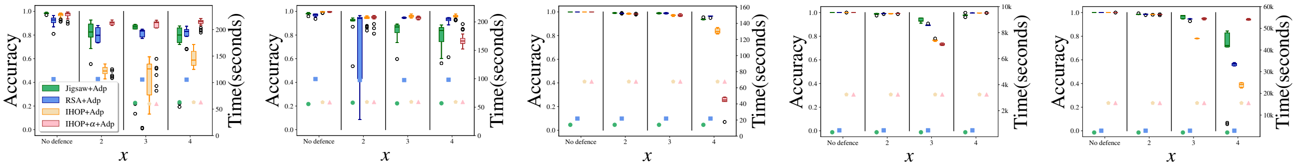
Against the cluster-based padding. Recall that the cluster-based padding [3, 43] first divides all the keywords into clusters with each containing no less than α keywords. Then, this countermeasure pads each keyword to the largest volume in its cluster. To improve the padding efficiency, we sort all keywords in ascending order based on their volume and assign each continuous α keywords to the same cluster. We set α to 2, 4, and 8 and present the results in Figure 14. We set the α in Jigsaw to 0.1 (considering the cluster-based padding that injects more noise to high-volume queries) and keep other parameters the same as in Section 7.

We observe that the average accuracy of all the attacks exceeds 75%. But Enron consists of fewer distinctive queries than other datasets, leading to exceptional cases in the performance. As keywords with similar volumes are assigned to the same cluster, they become indistinguishable in terms of volume, which negatively impacts the accuracy of Jigsaw's first module. Beyond that, the cluster-based padding additionally brings instability in attacks' performance (see outliers in the figure). However, the average accuracy still remains practical.



(a) Enron (b) Lucene (c) Wikipedia, $|W| = 1000$ (d) Wikipedia, $|W| = 3000$ (e) Wikipedia, $|W| = 5000$

Figure 14: Comparisons with RSA and IHOP against the cluster-based padding [3, 43] in Enron, Lucene, and Wikipedia.



(a) Enron (b) Lucene (c) Wikipedia, $|W| = 1000$ (d) Wikipedia, $|W| = 3000$ (e) Wikipedia, $|W| = 5000$

Figure 15: Comparisons with RSA and IHOP against the padding in SEAL [14] in Enron, Lucene, and Wikipedia.

Against the Padding in SEAL. We present the results against the padding in SEAL [14], which pads the volume of keywords to the nearest power of an integer x. We note again that our attack only targets the padding strategy in SEAL. We set x to 2, 3, and 4 for the padding and demonstrate the results in Figure 15. Note that other parameters remain consistent with those previously configured. As x grows, the average accuracy of Jigsaw maintains above 70%, although there are a few outliers in the results. The padding also affects the performance of RSA and IHOP, causing an unstable and dropping trend. We summarise that though the results contain outliers when against the padding in SEAL and the cluster-based padding, the tested attacks provide fine accuracy in most cases under our settings.