Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption

Simon Oya
University of Waterloo

Florian Kerschbaum
University of Waterloo

Abstract

Recent Searchable Symmetric Encryption (SSE) schemes enable secure searching over an encrypted database stored in a server while limiting the information leaked to the server. These schemes focus on hiding the access pattern, which refers to the set of documents that match the client's queries. This provides protection against current attacks that largely depend on this leakage to succeed. However, most SSE constructions also leak whether or not two queries aim for the same keyword, also called the *search pattern*.

In this work, we show that search pattern leakage can severely undermine current SSE defenses. We propose an attack that leverages both access and search pattern leakage, as well as some background and query distribution information, to recover the keywords of the queries performed by the client. Our attack follows a maximum likelihood estimation approach, and is easy to adapt against SSE defenses that obfuscate the access pattern. We empirically show that our attack is efficient, it outperforms other proposed attacks, and it completely thwarts two out of the three defenses we evaluate it against, even when these defenses are set to high privacy regimes. These findings highlight that hiding the search pattern, a feature that most constructions are lacking, is key towards providing practical privacy guarantees in SSE.

1 Introduction

Searchable Symmetric Encryption (SSE) [6] is a type of private search that allows a client to store an encrypted database in a server while being able to perform searches over it. In a typical SSE scheme, the client first encrypts the database using private-key encryption, generates a search index, and sends them to the server. Then, the client can perform queries by generating query tokens, that the server evaluates in the index to obtain which documents match the query.

To appear in: Proceedings of the 30th USENIX Security Symposium
August 11–13, 2021, Vancouver, B.C., Canada
https://www.usenix.org/conference/usenixsecurity21

There are different types of private search techniques that provide different security guarantees and query functionalities, such as range or SQL queries. Fuller et al. [10] give an overview of protected search schemes and examples of companies that offer products with searchable encryption. In this work, we focus on point queries, which are the main query type in SSE schemes. Namely, we consider that each document in the database has a list of *keywords* associated with it, and the client queries for documents that match a certain keyword. The typical use case of keyword searches in related work are email databases [3, 15, 22, 26, 28].

Even though the database and the query tokens are encrypted, basic SSE schemes leak certain information to the server when performing a query. There are two main sources of leakage considered in the literature: the *access pattern*, which refers to the identifiers of the documents that match a query; and the *search pattern*, also known as query pattern, which refers to identifying which queries in a sequence are identical. An honest-but-curious server can leverage this leakage to identify the client's queries (query recovery attacks) or the database contents (database recovery attacks).

Liu et al. [22] proposed one of the few attacks that exploits only search pattern leakage to recover queries. The search pattern allows the adversary to compute the *frequency* with which the client performs each query. After observing queries for a long time, the attacker can compare the frequency information of each query token with auxiliary data to identify each query's keyword. Islam et al. [15] proposed an attack (IKK) that leverages keyword co-occurrence information extracted from the access pattern leakage, as well as certain ground truth information about the client's queries, to identify the remaining queries. Further refinements of this idea improve the attack when the keyword universe is large [3] and even allow the adversary to infer the keywords without ground truth and with imperfect auxiliary information [26].

In order to protect the client against these attacks, the research community has proposed *privacy-preserving SSE schemes* with reduced leakage. Schemes that completely hide the search pattern, such as those based on Oblivious RAM

(ORAM) [11], require running a protocol with a typically prohibitive communication cost. Also, they still leak the *response volume*, i.e., how many documents are returned in response to a query, which can be exploited by certain attacks [3].

Recent proposals trade in communication or computational efficiency for privacy. Some of these defenses propose relaxations of the notion of ORAM [7], or simply obfuscate the access pattern by adding false positives and false negatives to the set of documents that match a query [4]. Recent work by Patel et al. [24] proposes using hashing techniques to completely obfuscate the access pattern structure, and hide the response volume by padding it with Laplacian noise.

The privacy guarantees of these and other defenses can be assessed theoretically or empirically. Theoretical notions include the differential privacy framework [8], used to protect access pattern leakage [4] or response volume [24], or quantifying the number of information bits revealed per query [7]. The problem with these theoretical notions is that it is hard to judge how well they translate into actual protection guarantees against attacks. Assessing the performance of defenses empirically using generic SSE attacks can however overestimate the protection of these defenses. Most works either evaluate their proposals against ad-hoc attacks [7], figure out how to extend existing attacks to a given defense (e.g., Chen et al. [4] adapt IKK [15]), or simply rely only on a theoretical guarantee [24]. The effectiveness of current defenses has only been evaluated against attacks that exploit access pattern leakage, while search pattern leakage has only recently been explored in the particular case of range and nearest-neighbor queries [19].

In this work, we aim at investigating to which extent leaking the search pattern affects the privacy of SSE schemes that allow point queries. In order to achieve this, we propose the first query identification attack that simultaneously combines access and search pattern leakage, as well as some auxiliary (background) information, to identify the keywords of the client's queries. We note that, even though certain attacks rely on strong background information [3, 15] to achieve high accuracy [2], our assumptions on background information are weak. For example, we do not assume that the adversary knows the true distribution of the documents/keywords nor any ground-truth information. Instead of relying on heuristics, we develop our attack following a Maximum Likelihood Estimation (MLE) approach. This makes our attack easy to adapt against specific defenses, and we illustrate this by modifying our attack to perform well against three of the most recent privacy-preserving SSE schemes for point queries [4, 7, 24].

We compare our attack with the state-of-the-art graph matching attack by Pouliot and Wright [26], and show that our proposal is orders of magnitude faster and has a higher query recovery accuracy than graph matching when the client does not query for every possible keyword in the dataset. Our attack also outperforms one of the few attack that uses search pattern leakage [22]. The main reason that our attack outperforms previous works is that it *combines* volume and frequency leakage information. Our attack achieves 74%, 48%, 37%, and 22% query recovery rate for keyword universes of sizes 100, 500, 1 000, and 3 000, respectively, after observing only ≈ 250 (possibly repeated) queries from the client.

We tune our attack against three recent privacy-preserving SSE schemes [4, 7, 24] and evaluate its performance with two real datasets. Our experiments reveal that these defenses are highly effective against a naive attack that does not take the defense into account (e.g., lowering the accuracy with 1 000 possible keywords from 37% to 1.4%, 2.4%, and 2.7% respectively for defenses [4], [24], and [7], configured to high privacy regimes). When adapting our attack against the defenses, the accuracy increases back to 30%, 35%, and 23%, respectively. This shows that two of the defenses fail at achieving meaningful protection levels even though they incur more than 400% communication overhead. The third defense [7] is both more efficient and effective, but our attack still recovers a non-trivial amount of keywords against it.

To summarize, our contributions are:

1. We derive a new query recovery attack for SSE schemes following a maximum likelihood estimation approach. Our attack combines information from both access and search pattern leakage.

2. We evaluate our attack against a basic SSE scheme and show that it is more accurate than the state-of-the-art access pattern-based attack and one of the few attacks that relies exclusively on search pattern leakage.

3. We provide a methodology to adapt our attack against particular SSE defenses and illustrate our approach by tailoring our attack to perform well against three recent proposals.

4. We evaluate our attack against these three defenses and show that two of them in practice fail to protect the queries and we still recover a non-trivial amount of queries against the third one.

The rest of the paper is organized as follows. We summarize related work in the next section. In Section 3 we introduce our general leakage model for SSE schemes that we use to derive our attack in Section 4 and adapt it against defenses in Section 5. We compare our attack with others and evaluate it against SSE defenses in Section 6, discuss how to hide search pattern leakage in Section 7 and conclude in Section 8.

2 Related Work

Searchable Symmetric Encryption (SSE) [6] is one type of *protected search* technique. Other popular protected search techniques include Property-Preserving Encrpytion (PPE) [23] and Privacy Information Retrieval (PIR) [5]. We

refer to the SoK paper by Fuller et al. [10] for a thorough revision of these and other protected database search techniques. In this section, we summarize the main attacks and defenses in SSE, with a particular focus on point queries, which is the subject of our work.

2.1 Attacks against SSE Schemes

Attacks against SSE schemes can be broadly classified based on whether they consider an active or passive adversary, the type of queries allowed by the scheme, the leakage required by the attack, and the goal of the attack.

File injection attacks [3, 28] consider an *active adversary* that is able to insert documents in the database. This is reasonable, for example, if the database is an email dataset and the adversary can send emails to be stored in that dataset. By carefully choosing the keywords of the inserted documents and studying which of these files match a certain query, the adversary can identify the underlying keyword of such query.

We can broadly classify *passive attacks* according to their goal into database and query recovery attacks. Database recovery attacks aim to recover the content of the database, while query recovery attacks aim to find the target of each of the client's queries. In some schemes, query recovery attacks can be used to recover the contents of the database by checking which queries trigger a match for each document.

Database recovery is a typical goal of attacks in *range query* schemes. In these schemes, each document has a particular attribute value and the client can retrieve documents whose attribute is within a given range. Previous works study the complexity of recovering the attribute values in the dataset based on the access pattern leakage of range queries [13, 14, 18, 21]. Recent work by Kornaropoulos et al. [19] also uses the search pattern leakage (i.e., whether or not two queries are identical) to develop reconstruction attacks for range and k-nearest neighbor query schemes. These works are not necessarily relevant for our work, since they require schemes that allow range queries.

Query recovery is a typical goal of attacks against SSE schemes where the client performs *point queries*, i.e., it queries for the set of documents that contain a certain keyword. In this setting, we can generally distinguish between attacks that use access pattern leakage and those that use search pattern leakage.

The seminal work by Islam et al. [15] (known as IKK attack) shows that it is possible to recover the client's queries using *access pattern* leakage, but relies on strong assumptions on background information. In this attack, the adversary stores how many documents match every pair of distinct queries and compares this with auxiliary information about keyword co-occurrence. Then, it matches each received query with a keyword using a heuristic algorithm that also relies on ground truth information about a subset of the queries. Cash et al. [3] showed that IKK does not perform well when the subset of possible keywords is large (e.g., 2 500 keywords) and propose an alternative attack that identifies keywords based on their response volume (i.e., the number of documents that match the query). The most recent iteration of these attacks, by Pouliot and Wright [26], proposes a graph matching attack that allows the adversary to accurately recover the queries even when the adversary has imperfect auxiliary information about the statistical distribution of the dataset.

The attack proposed by Liu et al. [22] relies only *search pattern* leakage. This attack assigns a tag to each distinct query it receives, and uses the search pattern leakage to monitor the frequency of each tag over time. Then, the adversary can recover the underlying keyword of each tag by comparing the tag query trends with keyword trend information.

Ours is the first attack against SSE schemes where the client performs point queries that leverages *both access and search pattern* leakage. Our attack takes core ideas from related works [22, 26], but relies on a Maximum Likelihood Estimation (MLE) approach to find the most likely keyword of each received query. The techniques we use to solve our attack are somewhat similar to the frequency-based database recovery attacks by Bindschaedler et al. [1] in deterministic encryption. However, our adversary model is conceptually very different since it aims at query recovery, and our attack leverages both frequency and volume (search pattern) information.

2.2 Privacy-Preserving SSE Schemes

Early works that introduce attacks against SSE schemes also propose the first techniques to partially hide access pattern information [15] or query frequencies [22] to palliate the effects of these attacks. Even though one can build protected search techniques based on Oblivious RAM (ORAM) [12] that completely hide the search pattern (and possibly the access pattern), such as TwoRAM [11], their practicality is still questionable since they incur a significant communication overhead and they still leak the query volume information. Kamara et al. [17] provide a framework to design structured encryption schemes while hiding the access and search pattern. Their approach is based on the square-root ORAM by Goldreich and Ostrovsky [12], and introduces the notion of volume-hiding encrypted multimap schemes to hide the volume information (e.g., how many documents are associated with every search key). Patel et al. [24] propose more efficient volume-hiding techniques. They explain why completely hiding the query response volume is unreasonably expensive, and introduce differentially-private volume-hiding, which trades leakage for efficiency.

Chen et al. [4] propose a framework to hide access patterns in a differentially private way. In their scheme, the client first generates an inverted index, i.e., a structure indicating which documents contain which keywords, and obfuscates it by adding false positives and false negatives. This obfuscation

adds noise to the access patterns and thus makes it harder to apply attacks such as IKK [15] against it. They palliate false positives by using a document redundancy technique.

Finally, recent work by Demertzis et al. [7] proposes an ORAM-based scheme with the idea of hiding bits of information about the address of a document in the database and the response volume of a query. For this, they split the dataset into 2^{α} ORAM blocks that hide which document within the block is accessed each time, and pad the response volume of each query to the next power of a constant x. The values of α and x allow to adjust the privacy vs. utility trade-off of this scheme.

3 Preliminaries

We consider a client-server scenario where the client owns a database and, for the sake of saving storage space, wants to outsource it to the server while keeping the ability to perform *point* queries over it. The client uses a (privacy-preserving) SSE scheme for this, that works as follows. First, the client encrypts the database using symmetric encryption and sends it to the server, together with a query index. Then, when the client wants to query for a particular keyword, it generates a query token and sends it to the server. The server evaluates the query token on the index and obtains the addresses of the documents that match the query. The server returns these documents to the client. The client wants to keep both the underlying keyword of each query and the contents of the database secret (keyword and database privacy).

The adversary that we consider is an honest-but-curious server that follows the protocol but might use the information it observes to infer private information. Throughout the text, we refer to the server as adversary or attacker. We focus on query recovery attacks, i.e., the goal of the adversary is to identify the underlying keyword behind each query. In some cases, the adversary can leverage query recovery attacks to recover the database by identifying the set of keywords that trigger a match for each document in the database. We always assume that the adversary knows the parameters and algorithms of the SSE scheme, following Kerckhoffs' principle.

3.1 System Model and Notation

We present a general model that captures the leakage of many proposed privacy-preserving SSE schemes while abstracting from the cryptographic and implementation details of these protocols. The notation that we use is summarized in Table 1. We use upper-case boldface characters to denote matrices and lower-case boldface characters to denote vectors. The (i,j)th entry of matrix $\mathbf{A}$ is $(\mathbf{A})_{i,j}$, and $\mathrm{tr}(\mathbf{A})$ is the trace of $\mathbf{A}$. We represent the natural logarithm as log; other logarithm bases are written explicitly.

Let $\Delta = [w_1, w_2, \ldots, w_n]$ be the keyword universe, where w_i is the ith keyword, and let $n \doteq |\Delta|$ be the total number of keywords. Let N_D be the number of documents in the encrypted database that the client sends to the server. For each query, the adversary observes the tuple $(t, \mathbf{a})$ where t is the timestamp of the query and $\mathbf{a}$ is the *access pattern*, i.e., a vector with the positions of the documents that match the query. The leakage of all the SSE schemes that we consider in this work can be characterized by a sequence of tuples $(t, \mathbf{a})$. We use $|\mathbf{a}|$ to denote the *response volume*, i.e., the number of documents returned to the client in response to a query.

We consider SSE schemes that leak the *search pattern*, i.e., they leak which queries within a sequence are for the same keyword. The search pattern leakage can be explicit or implicit. Explicit search pattern occurs when querying for a certain keyword always generates the same query token [4,6,24]. Implicit leakage refers to SSE schemes where the queries for the same keyword w_i always generate the same access pattern $\mathbf{a}$, and the adversary can compare access patterns to check whether or not different tokens aim for the same keyword [7]. We discuss how to hide search patterns in Section 7.

Using the search pattern leakage, the adversary can assign a *tag* to each different access pattern it observes. The number of tags m will be at most equal to the number of keywords n (i.e.,

	General Parameters
Δ	Keyword universe $\Delta \doteq [w_1, w_2, \ldots, w_n]$.
n	Total number of keywords, $n \doteq \lvert\Delta\rvert$.
w_i	ith keyword, with $i \in [n]$.
N_D	Number of documents in the encrypted dataset.
ρ	Number of observation time intervals.
	Adversary Observations
m	Number of tags (distinct access patterns observed).
γ_j	jth tag, with $j \in [m]$.
$\mathbf{a}_j$	Access pattern assigned to tag j.
v_j	Volume of a query with tag j, $v_j \doteq \lvert\mathbf{a}_j\rvert$.
$\mathbf{v}$	Volume of tags, $\mathbf{v} \doteq [v_1, \ldots, v_m]$.
$\mathbf{M}$	Tag co-occurrence matrix (size $m \times m$).
η_k	Number of queries sent in the kth time interval.
$\boldsymbol{\eta}$	Vector $\boldsymbol{\eta} \doteq [\eta_1, \eta_2, \ldots, \eta_\rho]$.
$f_{j,k}$	Query frequency of γ_j in the kth time interval.
$\mathbf{f}_j$	Query frequency vector of γ_j, $\mathbf{f}_j \doteq [f_{j,1}, \ldots, f_{j,\rho}]$.
$\mathbf{F}$	Query frequency matrix of all tags (size $m \times \rho$).
	Auxiliary (Background) Information
$\tilde{v}_i$	Auxiliary volume information for keyword w_i.
$\tilde{\mathbf{v}}$	Volume vector of keywords, $\tilde{\mathbf{v}} \doteq [\tilde{v}_1, \ldots, \tilde{v}_n]$.
$\tilde{\mathbf{M}}$	Auxiliary keyword co-occurrence matrix ($n \times n$).
$\tilde{f}_{i,k}$	Query frequency of w_i in the kth time interval.
$\tilde{\mathbf{f}}_i$	Query frequency vector of w_i, $\tilde{\mathbf{f}}_i \doteq [\tilde{f}_{i,1}, \ldots, \tilde{f}_{i,\rho}]$.
$\tilde{\mathbf{F}}$	Query frequency matrix of all keywords (size $n \times \rho$).
	Attack Goal
$p(j)$	Index of the keyword that the attack assigns to γ_j.
$\mathbf{P}$	Permutation matrix, $\mathbf{P}_{p(j),j} = 1$, else 0 ($n \times m$).

Table 1: Summary of notation

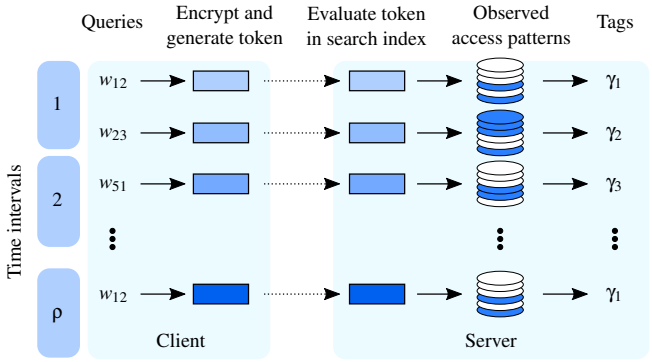


Figure 1: System Model

$m \leq n$), and will be strictly smaller if the client does not query for all possible keywords during the observation time. We use $\mathbf{a}_j$ to denote the access pattern of the jth tag, with $j \in [m]$. Then, the goal of the query recovery attack is to assign each tag its correct keyword. We denote this assignment, which is an injective mapping, by $p(\cdot) : [m] \to [n]$. We also represent it in matricial form as a $(n \times m)$ permutation (column-selection) matrix that we denote by $\mathbf{P}$ and define as

$$(\mathbf{P})_{i,j} = \begin{cases} 1\,, & \text{if } i = p(j)\,, \\ 0\,, & \text{otherwise.} \end{cases} \tag{1}$$

Figure 1 illustrates this model and notation. In the figure, the client queries for keywords $w_{12}, w_{23}, w_{51}, \ldots, w_{12}$. The server evaluates the query tokens in the search index and obtains which documents in the encrypted database match each query (i.e., the observed access patterns). Then, the server assigns a tag γ_j to each distinct access pattern. Note that the access patterns that result from evaluating different query tokens generated from the same keyword (e.g., w_{12}) are identical. The goal of the attack is to map each γ_j to a keyword w_i. In order to perform this mapping, the server uses information from the structure of the access patterns and from the frequency with which the server observes each access pattern, as well as some auxiliary information that we specify below.

Below, we define different data structures that the adversary can compute from the observations. Several query recovery attacks [15, 22, 26], as well as our proposal, can be defined by using these variables. The following structures are computed from the access patterns:

- **Query volume** ($\mathbf{v}$, v_j). The query volume refers to the number of documents in the database that are returned as a response to a certain query. We use $v_j \in [0,1]$ to denote the normalized volume of the jth tag, i.e., $v_j \doteq |\mathbf{a}_j|/N_D$, and $\mathbf{v} \doteq [v_1, \ldots, v_m]$.
- **Co-occurence matrix** ($\mathbf{M}$). This variable refers to the number of documents that simultaneously match two different queries, normalized by the total number of documents in the database. We use $\mathbf{M}$ to denote the symmetric matrix whose (i,j)th element is $(\mathbf{M})_{i,j} \doteq |\mathbf{a}_i \cap \mathbf{a}_j|/N_D \in [0,1]$.

The following structures are computed from the search patterns, i.e., from how many times the client sends a query tagged as γ_j. In order to compute these structures, the adversary first splits the observation time into ρ intervals (e.g., weeks).

- **Query number** ($\boldsymbol{\eta}$, η_k). We use η_k to denote the number of queries the client sent in the kth interval, and define the vector $\boldsymbol{\eta} \doteq [\eta_1, \ldots, \eta_\rho]$.
- **Query frequency** ($\mathbf{F}, \mathbf{f}_j, f_{j,k}$). The query frequency refers to how often the client performs a certain query. For each tag γ_j ($j \in [m]$) and each time interval, indexed by $k \in [\rho]$, we use $f_{j,k}$ to denote the frequency of tag j in the kth interval, i.e., the total number of times the client queries for tag j in the interval, divided by the total number of queries in that interval. We use $\mathbf{f}_j$ to denote the vector that stores $f_{j,k}$ for all $k \in [\rho]$ and $\mathbf{F}$ is the $(m \times \rho)$ matrix that stores all the frequencies.

In addition to the observations, the adversary has certain *auxiliary background information* (e.g., a training set) that helps them carrying out the query recovery attack. The adversary uses this information to compute data structures like the ones defined above, but for each keyword instead of each tag. We denote the auxiliary query volume information by $\tilde{v}_i$ for each keyword $i \in [n]$, the $n \times n$ co-occurrence matrix of keywords by $\tilde{\mathbf{M}}$, and the $n \times \rho$ matrix storing the query trends of each keyword by $\tilde{\mathbf{F}}$. We note that background information is a strong assumption and attacks that rely on high-quality auxiliary information to be effective might be unrealistic [2]. In our evaluation in Section 6, we show that our attack is strong under weak assumptions on the auxiliary information. Namely, in our experiments the adversary computes $\tilde{\mathbf{v}}$ and $\tilde{\mathbf{M}}$ using a training set that is disjoint with the actual client's database, and $\tilde{\mathbf{F}}$ using public information about query trends with a time offset.

Below, we explain state-of-the-art query recovery attacks using access pattern [26] and search pattern [22] leakage using our notation.

3.2 Graph Matching Attack

In the graph matching attack by Pouliot and Wright [26], the adversary represents the set of tags and the set of keywords as two graphs, and the goal is to solve a *labeled graph matching* problem between the graphs. Let the keyword graph be $\tilde{G}$ (it has n nodes), and let the tag graph be G (it has m nodes). The labeled graph matching problem looks for the permutation

matrix $\mathbf{P}$ that minimizes the convex combination of two objective functions that measure a similarity score between the graphs.

The first objective function is based on the adjacency matrices of each graph, that determine the weights of the edges between nodes. The adjacency matrix of $\tilde{G}$ is $\tilde{\mathbf{M}}$, and the adjacency matrix of G is $\mathbf{M}$. Given an assignment of keywords to tags $\mathbf{P}$, the adjacency matrix of an upscaling of G to match the size of $\tilde{G}$ would be $\mathbf{PMP}^T$. Therefore, it makes sense to look for the permutation $\mathbf{P}$ that minimizes

$$||\tilde{\mathbf{M}} - \mathbf{PMP}^T||_F^2 , \tag{2}$$

where $||\cdot||_F$ denotes the Frobenius norm of matrices.[1]

Additionally, the labeled graph matching attack considers another objective function that depends only on the volume of each keyword/tag. The attack builds a $n \times m$ similarity matrix $\mathbf{C}$ whose (i,j)th element measures the likelihood of the assignment of γ_j to keyword w_i. Pouliot and Wright [26] compute this likelihood assuming that the number of matches of a certain keyword w_i in the encrypted dataset follows a Binomial distribution with N_D trials (dataset size) and a match probability given by the volume of that keyword in the auxiliary information $\tilde{v}_i$. Then, the (i,j)th element of $\mathbf{C}$ is

$$(\mathbf{C})_{i,j} = \binom{N_D}{N_D v_j} \cdot \tilde{v}_i^{N_D v_j} (1-\tilde{v}_i)^{N_D(1-v_j)} . \tag{3}$$

It then makes sense to maximize the trace $\mathrm{tr}(\mathbf{P}^T\mathbf{C})$.

Putting all together, the attack solves the problem

$$\mathbf{P} = \underset{\mathbf{P}\in\mathcal{P}}{\mathrm{argmin}} \quad (1-\alpha)\cdot||\tilde{\mathbf{M}} - \mathbf{PMP}^T||_F^2 - \alpha\cdot\mathrm{tr}(\mathbf{P}^T\mathbf{C}) , \tag{4}$$

where α is the coefficient of the convex combination that the attacker must tune in order to optimize its performance. Here, we have used $\mathcal{P}$ to denote the set of all valid column-selection permutation matrices $\mathbf{P}$.

The algorithms in the package[2] used by Pouliot et al. [26] to run this attack only work when the graphs have the same number of nodes, i.e., $m = n$, which is almost never the case in practice. When $m < n$, by default the package fills the smallest graph with dummy nodes (e.g., it adds zeros to $\mathbf{M}$). We show in Section 6 that this hampers the performance of the attack when $m \ll n$.

3.3 Frequency Attack

We explain the basic frequency attack by Liu et al. [22]. In this attack, the adversary builds the frequency matrix for the tags $\mathbf{F}$, and uses the frequency matrix for keywords $\tilde{\mathbf{F}}$ as auxiliary-information. The attacks assigns the keyword w_i to tag γ_j as

$$p(j) = \underset{i\in[n]}{\mathrm{argmin}} ||\mathbf{f}_j - \tilde{\mathbf{f}}_i||_2 , \tag{5}$$

where $||\cdot||_2$ the Euclidean norm for vectors. The attack simply chooses, for each tag γ_j, the keyword w_i whose frequency trend ($\tilde{\mathbf{f}}_i$) is closest in Euclidean distance to the trend information of the tag ($\mathbf{f}_j$). This decision is independent for each tag, so several tags can be mapped to the same keyword (i.e., $p(\cdot)$ is not injective).

Liu et al. also propose a more complex attack for a different query model where the client has preference for querying for keywords of a certain semantic category, and the adversary does not know this category a-priori. We do not consider this setting in our work, for generality.

4 Search and Access Pattern-Based Query Recovery Attack

We develop a query recovery attack that combines ideas from previous works [22, 26], but follows a pure Maximum Likelihood Estimation (MLE) approach and is orders of magnitude faster than the graph matching attack [26]. In particular, we look for the mapping $\mathbf{P}$ that maximizes the likelihood of observing $\mathbf{v}$, $\mathbf{F}$, $\boldsymbol{\eta}$ and N_D given the auxiliary information $\tilde{\mathbf{v}}$ and $\tilde{\mathbf{F}}$. We deliberately decide not to use the co-occurrence matrices $\mathbf{M}$ and $\tilde{\mathbf{M}}$ to help us estimate $\mathbf{P}$, for two reasons. First, certain SSE techniques already hide keyword co-occurrence information [7, 24], as Blackstone et al. [2] explain. Second, it might be hard to obtain auxiliary keyword co-occurrence information $\tilde{\mathbf{M}}$ that is close to the actual data co-occurrence $\mathbf{M}$. Our attack only uses background information from keyword volume $\tilde{\mathbf{v}}$ and frequencies $\tilde{\mathbf{F}}$, which in many use cases can be easily obtained (e.g., from statistics about English word usage).

Formally, our attack solves the maximum likelihood problem

$$\mathbf{P} = \underset{\mathbf{P}\in\mathcal{P}}{\mathrm{argmax}} \Pr(\mathbf{F}, \boldsymbol{\eta}, \mathbf{v}, N_D | \tilde{\mathbf{F}}, \tilde{\mathbf{v}}, \mathbf{P}) . \tag{6}$$

Note that it is not possible to exactly characterize this probability in practice. Instead, we rely on a *mathematical model* to characterize it. We emphasize that there is no "correct model" for this task, but models that are close to the actual semantic properties of the database and the client's querying behavior will yield more accurate estimates of the true $\mathbf{P}$, while very unrealistic models will produce estimates with poor accuracy. We use this mathematical model to derive our attack, and evaluate the performance of our attack with real data in Section 6.

4.1 Modeling the Observations

We aim at characterizing $\mathbf{F}$, $\boldsymbol{\eta}$, $\mathbf{v}$, and N_D given $\tilde{\mathbf{F}}$, $\tilde{\mathbf{v}}$, and an assignment of tags to keywords $\mathbf{P}$. We assume that the client's

[1] The original attack [26] considers the Frobenius (or Euclidean) norm, but the software package that they use to solve the problem [27] uses the Frobenius norm *squared*.

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

querying behavior and the response volumes are independent, i.e.,

$$\Pr(\mathbf{F},\boldsymbol{\eta},\mathbf{v},N_D|\tilde{\mathbf{F}},\tilde{\mathbf{v}},\mathbf{P}) = \Pr(\mathbf{F},\boldsymbol{\eta}|\tilde{\mathbf{F}},\mathbf{P})\cdot\Pr(\mathbf{v},N_D|\tilde{\mathbf{v}},\mathbf{P}) \quad (7)$$

In our model, the number of queries the client makes in each time interval, $\boldsymbol{\eta}$, follows an arbitrary distribution (independent of $\mathbf{P}$) that we represent as $\Pr(\boldsymbol{\eta})$. The client chooses the keyword of each query *independently* from other queries following the query frequencies $\tilde{\mathbf{F}}$. This means that the number of queries for each keyword $i \in [n]$ in time interval $k \in [\rho]$ follows a *Multinomial distribution* with η_k trials and probabilities given by $\tilde{\mathbf{f}}_k$. Formally,

$$\Pr(\mathbf{F},\boldsymbol{\eta}|\tilde{\mathbf{F}},\mathbf{P}) = \Pr(\boldsymbol{\eta})\cdot\Pr(\mathbf{F}|\tilde{\mathbf{F}},\boldsymbol{\eta},\mathbf{P}) \quad (8)$$

$$= \Pr(\boldsymbol{\eta})\cdot\prod_{k=1}^{\rho}\Pr(\mathbf{f}_k|\tilde{\mathbf{f}}_k,\eta_k,\mathbf{P}) \quad (9)$$

$$= \Pr(\boldsymbol{\eta})\cdot\prod_{k=1}^{\rho}\eta_k!\prod_{j=1}^{m}\frac{(\tilde{f}_{p(j),k})^{\eta_k f_{j,k}}}{(\eta_k f_{j,k})!}\,. \quad (10)$$

In our model, the number of documents in the encrypted database, N_D, is independent of $\mathbf{P}$, and the keywords of each encrypted document are chosen independently. More precisely, given the relative volumes of the keywords from the auxiliary information $\tilde{\mathbf{v}} = [\tilde{v}_1,\ldots,\tilde{v}_n]$, each document has keyword $i \in [n]$ with probability $\tilde{v}_i$. This implies that the response volume when the client queries for w_i will be a Binomial random variable with N_D trials and probability $\tilde{v}_i$, as in (3). Formally,

$$\Pr(\mathbf{v},N_D|\tilde{\mathbf{v}},\mathbf{P}) = \Pr(N_D)\cdot\Pr(\mathbf{v}|\tilde{\mathbf{v}},N_D,\mathbf{P}) \quad (11)$$

$$= \Pr(N_D)\cdot\prod_{j=1}^{m}\binom{N_D}{N_D v_j}\tilde{v}_{p(j)}^{N_D v_j}(1-\tilde{v}_{p(j)})^{N_D(1-v_j)}\,. \quad (12)$$

4.2 Maximum Likelihood Estimator

We use this model to find the $\mathbf{P}$ that maximizes $\Pr(\mathbf{F},\boldsymbol{\eta},\mathbf{v},N_D|\tilde{\mathbf{F}},\tilde{\mathbf{v}},\mathbf{P})$. We choose to maximize the logarithm of this probability instead to avoid precision issues (the problems are equivalent). We can ignore the additive terms in the objective function that are independent of $\mathbf{P}$, since they do not affect the optimization problem. The logarithm of equation (7) consists of two summands. The first one is the logarithm of (10). The only term that depends on $\mathbf{P}$ here is

$$\sum_{k=1}^{\rho}\sum_{j=1}^{m}\eta_k f_{j,k}\cdot\log(\tilde{f}_{p(j),k})\,. \quad (13)$$

The second term of (7) is (12). We can disregard $\Pr(N_D)$ and $\prod_{j=1}^{m}\binom{N_D}{N_D v_j}$ since they do not depend on $\mathbf{P}$, and the remainder is:

$$\sum_{j=1}^{m}\left[N_D v_j\log\tilde{v}_{p(j)} + N_D(1-v_j)\log(1-\tilde{v}_{p(j)})\right] \quad (14)$$

We can write the problem of maximizing the summation of (13) and (14) in matricial form as follows. First, we define two $n\times m$ cost matrices $\mathbf{C}_f$ and $\mathbf{C}_v$ whose (i,j)th entries are

$$(\mathbf{C}_f)_{i,j} \doteq -\sum_{k=1}^{\rho}\eta_k f_{j,k}\cdot\log(\tilde{f}_{i,k})\,, \quad (15)$$

$$(\mathbf{C}_v)_{i,j} \doteq -\left[N_D\cdot v_j\cdot\log\tilde{v}_i + N_D(1-v_j)\cdot\log(1-\tilde{v}_i)\right]. \quad (16)$$

We add a negative sign to these matrices so that we can formulate the maximization problem in (7) as an *unbalanced assignment problem*:

$$\mathbf{P} = \operatorname*{argmin}_{\mathbf{P}\in\mathcal{P}}\operatorname{tr}(\mathbf{P}^T(\mathbf{C}_v+\mathbf{C}_f))\,. \quad (17)$$

This problem can be efficiently solved with the Hungarian algorithm [20], whose complexity in the unbalanced case can be reduced to $O(n\cdot m + m^2\cdot\log m)$ as reported in [9].

Weighted Estimation. Sometimes, the adversary knows that their auxiliary volume information is more reliable than their frequency information, or vice-versa. In these cases, it might make sense to assign more weight to their relative contribution to the optimization problem in (17). The adversary can do this by considering a combination coefficient $\alpha\in[0,1]$ and define the objective function as

$$\mathbf{P} = \operatorname*{argmin}_{\mathbf{P}\in\mathcal{P}}\operatorname{tr}(\mathbf{P}^T[(1-\alpha)\mathbf{C}_v+\alpha\mathbf{C}_f])\,. \quad (18)$$

5 Adapting the Attack against Privacy-Preserving SSE Schemes

So far, we have considered a generic SSE scheme that does not hide the access and query patterns. This allows the adversary to compute the actual volume and frequency information, and carry out an attack with high accuracy (if the auxiliary information is accurate). While there are no efficient techniques to hide the search patterns, there are many proposals that obfuscate the access patterns and/or response volumes. In order to correctly assess the protection of these defenses, it is important to consider an attack performed by an adversary that is aware of the defenses implemented by the client.

In this section, we explain how to modify our attack to target particular privacy-preserving SSE schemes. We adapt the attack by characterizing the probability of each keyword response volume given the auxiliary information, $\Pr(\mathbf{v}|\tilde{\mathbf{v}},N_D,\mathbf{P})$, when the defense takes place. Following, we adapt the attack to three known privacy-preserving SSE schemes [4, 7, 24] that (partially) hide the access patterns, but our methodology applies to other existing (and future) defenses. We introduce only the minimum information about these defenses required to understand how to adapt our attack against them, and refer to their papers for more details. In Section 7 we briefly discuss how to use our attack when the SSE scheme also hides search patterns.

5.1 Differentially Private Access Patterns (CLRZ)

The SSE scheme by Chen et al. [4] (that we denote CLRZ) hides the access patterns by adding random false positives and false negatives to the inverted index of the database. This provides a certain level of indistinguishability between access patterns that can be expressed in terms of the differential privacy framework [8]. Let TPR and FPR be the true positive and false positives rates of the defense, respectively. First, the client generates an inverted index, i.e., a $N_D \times n$ binary matrix whose (ℓ, i)th element is 1 if the ℓth document has keyword w_i, and 0 otherwise. Then, each 0 in that matrix is flipped into a 1 with probability FPR, and each 1 is set to 0 with probability $1 - \texttt{TPR}$. This obfuscated matrix is used to generate the search index and determines which documents match each query.

Therefore, a document will match keyword w_i if this keyword was in the index before the obfuscation (probability $\tilde{v}_i$) and the defense didn't remove it (TPR) or if the keyword was not in the original index $(1 - \tilde{v}_i)$, but the defense added it (FPR). This means that, after applying the defense, the probability that a document has keyword i is

$$\tilde{v}_i \cdot \texttt{TPR} + (1 - \tilde{v}_i) \cdot \texttt{FPR}. \tag{19}$$

We can adapt the attack against this defense by replacing $\tilde{v}_i$ in (16) by (19).

5.2 Differentially Private Volume (PPYY)

The defense by Patel et al. [24] (that we denote PPYY) assumes that the server stores independent document and keyword pairs (i.e., the server stores a copy of each document for each keyword this document has). The documents are stored in a hash table such that $H(w_i||k)$ points to the kth document that has keyword w_i, or to any random document if there are less than k documents with keyword w_i. When querying for keyword w_i, the client sends the hashes $H(w_i||1), H(w_i||2), \ldots, H(w_i||v)$ (for a certain volume v) and receives the documents in those positions of the hash table. Since the server is storing independent document-keyword pairs, queries for different keywords are completely uncorrelated and thus it is not possible to infer information from the access pattern structure (such as the co-occurrence matrix $\mathbf{M}$). However, the scheme must use a different volume for each keyword, since padding each keyword to the same volume is overly expensive.

Patel et al. propose to obfuscate the volume by adding Laplacian noise to it, plus a constant value to ensure that this extra volume is never negative. If the Laplacian noise plus constant is negative for a keyword, the scheme would be lossy, i.e., there would be false negatives when querying for that keyword.

Let ε be the privacy parameter of the scheme. Adding Laplacian noise with scale $2/\varepsilon$ ensures ε-differential privacy for the leaked volumes, i.e., for low values of ε (e.g., $\varepsilon < 1$) an adversary would not be able to distinguish between two keywords whose response volumes differ by a single document.

In order to ensure a negligible probability that Laplacian noise plus a constant is negative for any keyword, we follow the approach by Patel et al. [24]: The probability that at least one of n independent samples from $\text{Lap}(2/\varepsilon)$ is smaller than a constant $2t/\varepsilon$ is upper bounded by $n \cdot e^{-t}$. We want this probability to be negligible, so we set $n \cdot e^{-t} = 2^{-64}$ and find that $t = \log n + 64 \cdot \log 2$.

Therefore, if we use $\bar{v}_j$ to denote the *true volume* of keyword $w_{p(j)}$, and $\lceil \cdot \rceil$ denotes the ceiling function, the observed volume for tag γ_j would be

$$v_j = \bar{v}_j + \lceil \text{Lap}(2/\varepsilon) + 2(\log n + 64 \cdot \log 2)/\varepsilon \rceil. \tag{20}$$

We use the ceiling function since volumes need to be integers. Note that the overhead of this scheme increases with the number of keywords n, because the constant padding term needs to ensure that none of the keywords gets negative padding.

We use this expression directly to compute $\Pr(\mathbf{v}|\tilde{\mathbf{v}}, N_D, \mathbf{P})$. In this case, we cannot derive a closed-form expression for $\mathbf{C}_v$ and compute it as follows: for each $i \in [n]$, compute the convolution between the probability mass functions of $\text{Bino}(N_D, \tilde{v}_i)$ and $\text{Lap}(2/\varepsilon)$ shifted by constant $2(\log n + 64 \cdot \log 2)/\varepsilon$ and discretized with the ceiling function. Then, $(\mathbf{C}_v)_{i,j}$ is the value of the resulting function evaluated at v_j.

5.3 Multiplicative Volume Padding (SEAL)

The SEAL defense technique, proposed by Demertzis et al. [7], has two parameters, α and x. In SEAL, the server stores the database in 2^α ORAM blocks, so that it is not possible to tell which document within each block is accessed each time. This means that SEAL leaks *quantized* versions of the true access patterns. Additionally, SEAL pads the response volume of each query to the closest power of x.

Our attack uses the access patterns to identify whether or not two queries are distinct (i.e., to infer the search pattern). We note that it is possible to obfuscate the search pattern by choosing a small enough α to cause collisions in the quantized access patterns of different queries. However, we argue that this requires such a small value of α that might significantly affect the efficiency of SEAL, so we still consider that queries for distinct keywords generate distinct access patterns, and thus SEAL leaks the search pattern. Note that this is the case in the original work [7], since the authors use large values of α (that are close to $\log_2 N_D$).

Let $\bar{v}_j$ be the true volume of keyword $w_{p(j)}$ in the dataset. The observed volume when querying for this keyword in SEAL is $x^{\lceil \log_x \bar{v}_j \rceil}$. We compute $\mathbf{C}_v$ as follows: for each $i \in [n]$, compute the probability that $\text{Bino}(N_D, \tilde{v}_i)$ falls between each interval $(x^{k-1}, x^k]$ for $k \in [\lceil \log_x N_D \rceil]$. Denote this probability by $\text{Prob}(k, i)$. Then, $(\mathbf{C}_v)_{i,j}$ is $\text{Prob}(\lceil \log_x v_j \rceil, i)$.

6 Evaluation

In this section, we compare the performance of our attack with the graph matching attack by Pouliot and Wright [26] and the frequency attack by Liu et al. [22], and evaluate our attack against the three defenses we considered above [4, 7, 24]. We denote our attack by `sap` (search and access pattern-based attack) to distinguish it from `graphm` [26] and `freq` [22].

We use Python3.7 to implement our experiments[3] and run then in a machine running Ubuntu 16.04 in 64-bit mode using 32 cores of an Intel(R) Xeon(R) CPU (2.00GHz) with 256 GB of RAM. We use Scipy's implementation of the Hungarian algorithm to run our attack (i.e., to solve (17)).

Experimental Setup. We use two publicly available email datasets to build the client's database and the server's auxiliary information. The first dataset is Enron email corpus,[4] which contains 30 109 emails from Enron corporation, and is popular among related works [3, 15, 22, 26, 28]. The second dataset, used by Cash et al. [3], is the *java-user* mailing list from the lucene project.[5] We took the emails of this mailing list from September 2001 until May 2020 (around 66 400 emails). Each email is one document in the dataset, and its keyword list is the set of words in the main body of the email that are part of an English dictionary, excluding English stopwords. We use Python's NLTK corpus[6] to get a list of all English words and stopwords.

We select the 3 000 most frequent keywords to build a set Δ_{3000} for each dataset. Then, in each experiment run, given n, we generate the keyword universe Δ by randomly selecting n keywords from Δ_{3000}. In each experiment run, we perform a random keyword selection and a random split of the dataset; we use half of the documents as the actual client's dataset, and give the other half to the adversary to use as auxiliary information to compute $\tilde{\mathbf{v}}$ and $\tilde{\mathbf{M}}$.

We could not find any public database with actual user query information for either of the databases. This is a common problem when evaluating attacks that use query frequency, as observed by Liu et al. [22]. Therefore, we use query information from Google Trends[7] to generate client queries [22]. For each keyword in Δ_{3000}, we get its search popularity for the past 260 weeks (ending in the third week of May 2020). We store these popularity values in a 3000×260 matrix. In each experiment run, given a particular keyword universe Δ of size n, we take the popularity of each of those keywords in the last 50 weeks and store it in a $n \times 50$ matrix that we denote $\mathbf{F}^*$. Then, we normalize the columns of this matrix so that they add up to one. The observation time is always 50 weeks, and we vary the *average* number of queries per week ($\bar{\eta}$) that the client performs. We generate the actual number of queries that the client performs for keyword w_i in week k by sampling from a Poisson distribution with rate $\bar{\eta} \cdot f_{i,k}$, where $f^*_{i,k}$ is the (i,k)th element of $\mathbf{F}^*$.

Since giving the true frequency information to the adversary would be unrealistic, we give the adversary outdated frequency information instead. For a certain week offset τ, the adversary's auxiliary frequency information is $\tilde{f}_{i,k} = f^*_{i,k-\tau}$. Note that the observed frequencies $f_{j,k}$ will only approach $f^*_{i,k}$ as $\bar{\eta} \to \infty$. In most of our experiments, we set a very low number of average queries per week ($\bar{\eta} = 5$), so the information the adversary gets from the query frequencies is very limited. We think this approach is more realistic than giving the adversary frequencies perturbed with Gaussian noise [22].

We perform 30 runs of each of our experiments (in parallel), using a different random seed for each. This randomness affects the keyword selection, the dataset split, the query generation, and the defense obfuscation techniques. The attacks are deterministic. We measure the query recovery *accuracy*, which we compute by counting how many of the client's queries the attack recovers correctly and normalizing by the total number of queries (with possibly repeated keywords). For completeness, we also report the percentage of unique keywords recovered in each experiment in the Appendix.

6.1 Preliminary Experiments for Our Attack

We perform a preliminary experiment to observe the effect of the auxiliary information offset τ in `sap`. We perform the attack on Enron dataset using only frequency information, i.e., $\alpha = 1$ in (18), and show these results in Figure 2 for different sizes of the keyword universe n and average number of weekly queries $\bar{\eta}$. We see that the frequency information slowly degrades with the offset (we see a slight peak at 50 weeks when $n = 100$, since this is almost one year and some query behaviors repeat yearly). Also, the accuracy decreases with the keyword universe size n, since estimating the keyword of each query becomes harder when there are more possible keywords to choose from. We use an offset of $\tau = 5$ in the remainder of the evaluation, since most of our experiments are for $\bar{\eta} = 5$ and we see that the accuracy degradation stabilizes after that.

We carry out a second experiment to understand how `sap` benefits from both access and search pattern leakage. We set $\bar{\eta} = 5$ (average of 250 queries in total over 50 weeks) and vary $\alpha \in [0, 1]$. We show the attack's accuracy for different keyword universe sizes n in Figure 3. The lines are the average accuracy of the attacks, and the shades represent the 95% confidence interval. The results are qualitatively similar in both datasets, although it is slightly easier to identify keywords in Lucene. This experiment reveals that using either volume ($\alpha = 0$) or frequency ($\alpha = 1$) information alone provides low accuracy values (e.g., below 15% for $n = 1000$ in Enron). However, combining both types of information provides an

[3] Our code is available at https://github.com/simon-oya/USENIX21-sap-code

[4] https://www.cs.cmu.edu/~./enron/

[5] https://mail-archives.apache.org/mod_mbox/lucene-java-user/

[6] https://www.nltk.org/howto/corpus.html

[7] https://trends.google.com/trends

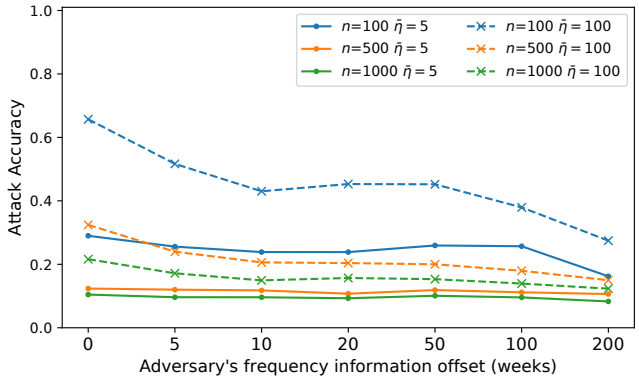


Figure 2: Effect of outdated frequency information in the performance of `sap` against a basic SSE in Enron dataset.

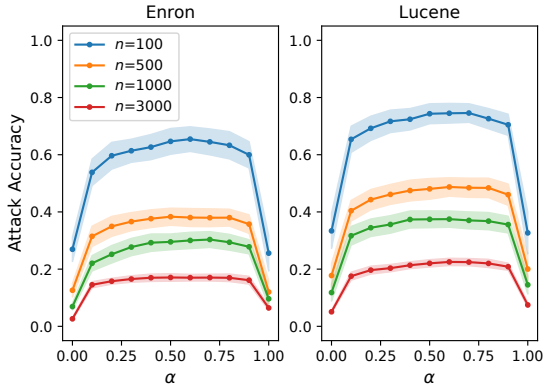


Figure 3: Effect of α in the performance of `sap` against a basic SSE ($\bar{\eta} = 5$ queries per week, 50 weeks).

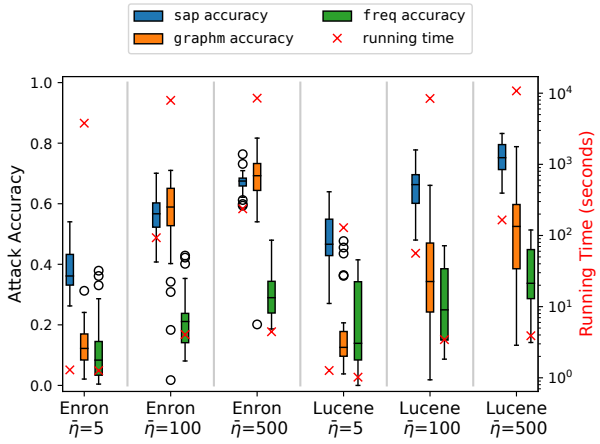


Figure 4: Comparison of the query recovery accuracy (boxes) and running time ($\times$) of attacks in different datasets with $\bar{\eta}$ queries per week (50 weeks), with $n = 500$ keywords.

outstanding boost (the accuracy is more than twice as large than when using either type of information by itself). In the remaining experiments, we use the pure maximum likelihood estimator ($\alpha = 0.5$) configuration for `sap`.

6.2 Comparison with Other Attacks

We compare the performance of `sap` with the graph matching attack by Pouliot et al. [26] (`graphm`) and the frequency attack by Liu et al. [22] (`freq`). We use the GraphM package[8] to solve the graph matching problem of `graphm`. This package offers different graph matching algorithms, and we use the PATH algorithm [27], since it provides the best results [26].

We show the results of our experiments in Figure 4. The boxes show the accuracy of the attacks (left axis), and the red crosses ($\times$) represent their average running time (right axis, logarithmic). We use the pure MLE approach for `sap` ($\alpha = 0.5$) and plot the results of `graphm` with the best performing α each time (we tried $\alpha = 0$ to $\alpha = 1$ with steps of 0.1). We use $n = 500$ for this plot (we do not use a larger number since the running times of `graphm` become unfeasible).

Our attack (`sap`) is approximately four times more accurate than `graphm` and `freq` when the client performs few queries ($\bar{\eta} = 5$) in both datasets. The performance of all the attacks increase as the adversary observes more queries, but `sap` takes the lead in most cases. For $\bar{\eta} = 500$ (a total of $\approx 25\,000$ queries observed), in Enron dataset, `graphm` achieves a slightly higher average accuracy than `sap`. However, note that the running time of `graphm` is always approximately two orders of magnitude larger than `sap` (note the logarithmic right axis).

Our experiments reveal that `graphm` heavily relies on observing almost all possible keywords to achieve high query recovery rates. We argue that this is a consequence of how the graph matching problem (4) is framed. Note that, when $m \ll n$, the matrix $\mathbf{PMP}^T$ will have many zero entries (the solver actually fills the smallest graph with dummy nodes, as we explain in Section 3.2). In this case, a good strategy to minimize (4) is to simply choose the permutation $\mathbf{P}$ that cancels the largest terms in $\tilde{\mathbf{M}}$. This permutation is not necessarily a good estimate of the the correct assignment of tags to keywords. This could potentially be solved by shrinking $\tilde{\mathbf{M}}$ instead, i.e., $||\mathbf{P}^T\tilde{\mathbf{M}}\mathbf{P} - \mathbf{M}||_F^2$ and/or using a norm that does not give more weight to large terms (e.g., opting for an $L1$-norm instead of the Frobenius or $L2$-norm). We note that improving this attack might still be unprofitable, since keyword co-occurrence is completely infective against recent SSE schemes [2].

In conclusion, the experiments confirm the relevance of our attack, since 1) it is computationally efficient, 2) it outperforms `freq`, 3) it outperforms `graphm` when the client does

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

not query for all possible keywords, which we argue is a realistic scenario. Also, our attack does not require background knowledge of keyword co-occurrence and is easily adaptable against defenses. This adaptability is key towards assessing the effectiveness of these defenses, as we show next.

6.3 Performance of `sap` against Defenses

We evaluate the performance of `sap` against the three defenses we considered in Section 5. We give the adversary the frequency information with an offset of $\tau = 5$ weeks and we set the observation time to 50 weeks, as before. The average number of queries per week is $\bar{\eta} = 5$ (i.e., average of 250 queries in total). We use this arguably low number to show that, even with a small number of queries, frequency information can really help the adversary. Again, we consider the pure MLE approach of `sap` (17), i.e., $\alpha = 0.5$. We evaluate the performance of the attack with up to $n = 3000$, since it is computationally efficient.

Performance against CLRZ [4]. We set the true positive rate of CLRZ to $\texttt{TPR} = 0.999$ and vary the `FPR` between 0.01, 0.05, and 0.1. Figure 5 shows the results in Enron (a) and Lucene (b). We generate the boxes using the accuracy values of `sap` in 30 runs of the experiment. The dotted black lines represent the mean accuracy of `sap` without adapting it against this defense, i.e., this would be the performance if the adversary was *unaware* of the defense. As a reference, the dotted blue lines show the performance of `sap` using frequency information only ($\alpha = 1$). The red crosses ($\times$) represent the bandwidth overhead of the defense (marked in the right axis), that we compute as follows. Let N_R be the total number of documents returned by the server in a run of the experiment, and let N_r be the number of documents that would be returned if the defense had not been applied. Then, the overhead percentage is $(N_R/N_r - 1) \cdot 100$. This value is only a reference, since the actual overhead depends on implementation details.

Increasing `FPR` improves the protection of the defense. For example, with $n = 1000$ keywords in Lucene, the attack accuracy drops from 37% (no defense) to $\approx 1\%$ ($\texttt{FPR} = 0.1$) against the naive attack (black doted line). However, by adapting the attack against the defense, the accuracy increases back to 30%. We observe this behavior in both datasets and for all values of n, which confirms that our attack is able to almost ignore the defense. Note that the maximum `FPR` value we consider ($\texttt{FPR} = 0.1$) indicates that around 10% of the whole dataset is returned in each query, which is already unrealistically high in real cases (the overhead is betwen 400% and 500% when $\texttt{FPR} = 0.1$).

Performance against PPYY [24]. We configure PPYY with privacy values $\varepsilon = 1$, 0.2, and 0.1. Note that smaller values of ε increase the amount of padding (and the overall privacy the scheme provides). Typically, in differential privacy scenarios, values of $\varepsilon < 1$ are considered high privacy regimes. Patel et al. [24] use $\varepsilon = 0.2$ in their cost evaluation.

Figure 6 shows the results in the same format as in the previous case. When computing the bandwidth overhead, we only take into account the overhead caused by the extra padding as explained above. The original scheme incurs extra overhead, e.g., due to the type of hashing technique used to store the database. We refer to their paper for the detailed cost analysis of this defense. Our goal with this experiment is to show the effectiveness of Laplacian noise as a volume-hiding technique.

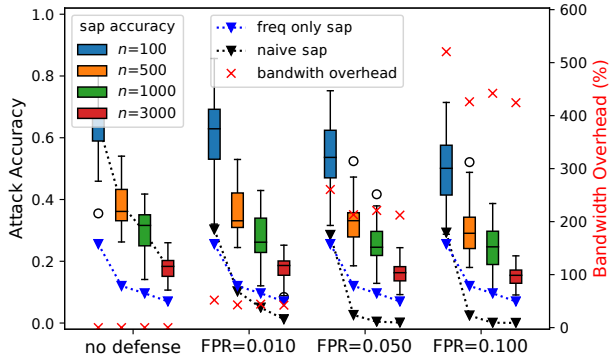
The results are qualitatively (and quantitatively) very close to the results for the previous defense. Values of $\varepsilon = 0.1$ seem to be effective at reducing the accuracy of the naive attack (dropping from 37% accuracy to $\approx 2\%$ in Lucene with $n = 1000$) but, when tailoring the attack against the defense, it recovers queries with a similar accuracy as when no defense is applied (35% in the aforementioned case).

The reason for this is the following: even though $\varepsilon = 0.1$ is a high differential privacy regime, this privacy notion only ensures that queries for keywords whose response volume differs *in one unit* are indistinguishable. As Patel et al. admit [24], in some settings this privacy definition might be unreasonable. This seems to be the case for the datasets we consider, and more generally it seems unrealistic to consider an optimistic setting where the only queries the adversary wants to distinguish are for keywords whose response volume differs in one document.

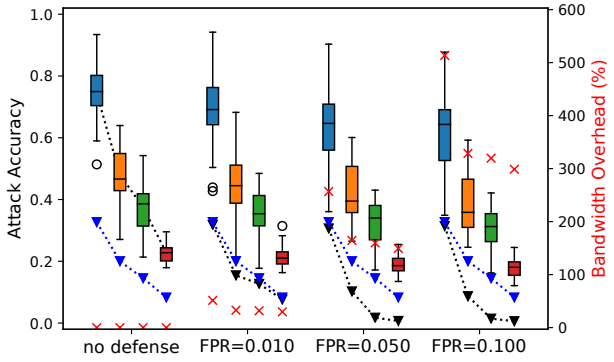
Performance against SEAL [7]. As we explain in Section 5.3, we assume that there are no collisions between the quantized access patterns that SEAL leaks, so that the scheme implicitly reveals the search pattern and the adversary can compute the query frequencies of each tag. We vary the multiplicative padding x between 2, 3, and 4. Recall that SEAL pads the volume of each keyword to the next power of x, and thus the overhead percentage is always smaller than $(x-1) \cdot 100$.

Figure 7 shows the results. Following the example above (Lucene with $n = 1000$), the attack accuracy drops from 37% to 3% with a padding parameter $x = 4$. A defense-aware attacker brings the accuracy up to 23%, which is still a significant value, but below the performance of the attack against the other two defenses. The results show that multiplicative volume padding is a highly efficient volume-hiding technique, since it achieves significantly more protection than the other two, with less bandwidth overhead.

We highlight that in all these experiments both the volume and the frequency information contribute the attack's success. This can be seen in the figures by noting that the boxes are significantly above the dashed blue lines (frequency-only `sap`).

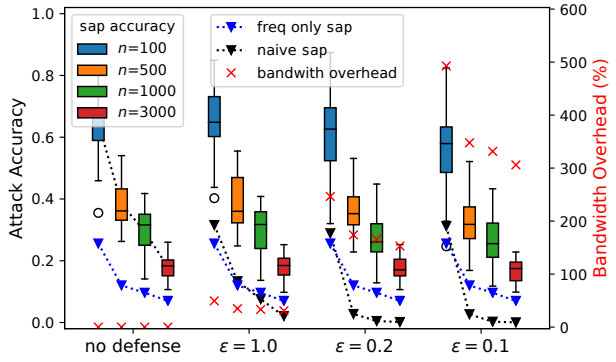


(a) Enron dataset

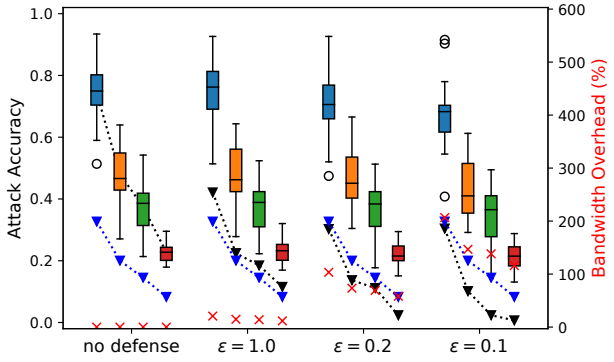


(b) Lucene dataset

Figure 5: Accuracy of `sap` against CLRZ defense configured with $\texttt{TPR} = 0.999$ and varying $\texttt{FPR}$ (50 weeks, $\bar{\eta} = 5$ queries/week).



(a) Enron dataset



(b) Lucene dataset

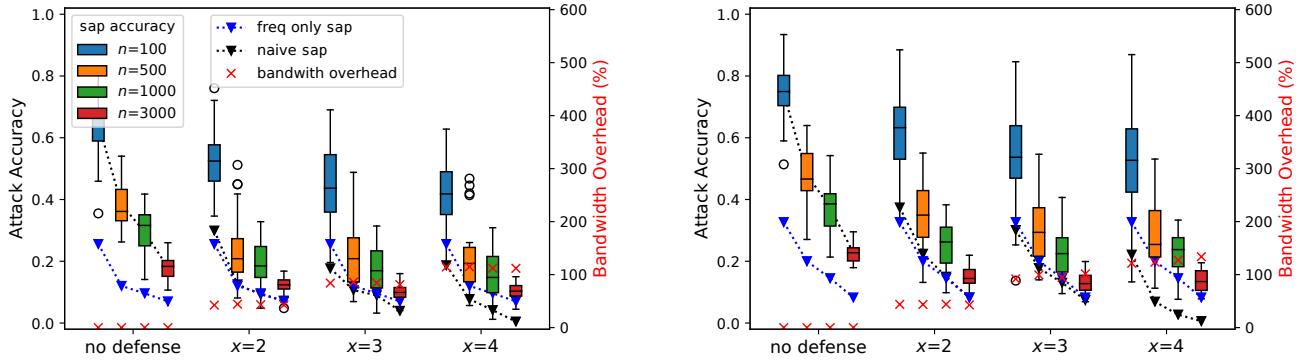
Figure 6: Accuracy of `sap` against PPYY defense with different privacy values ε (50 weeks, $\bar{\eta} = 5$ queries/week).

7 Discussion: Preventing Frequency Leakage

Throughout the paper, we have only considered defenses that obfuscate the access pattern and/or response volume. Completely hiding the volume information would require returning the same number of documents in response to every query, which is unreasonable in terms of bandwidth overhead [7, 16]. We have seen that, even when the volume is obfuscated, the frequency information (derived from the search pattern) surprisingly contributes to the success of our query identification attack. This is true even when the user only performs 5 queries per week and the observation time is 50 weeks (even if we consider keyword universes of size $n = 3000$). Below we discuss some alternatives for hiding this frequency information which we believe is key towards achieving effective privacy-preserving SSE schemes.

Hiding the Search Pattern with Collisions. Hiding the search pattern implies that the adversary is not able to tell whether or not a query has been repeated. This prevents the adversary from (correctly) assigning tags to queries and thus from computing observed query frequencies.

One option to hide the search pattern among groups of keywords is to create collisions between access patterns, i.e., force queries for different keywords to return the same set of documents. This idea of "merging keywords" is similar to the Secure Index Matrix [15] and, to some extent, to the Group-Based Construction [22]. In practice, it is still not clear how to provide privacy by grouping keywords while keeping the overhead of the scheme under reasonable bounds. This is because it is more efficient to merge keywords that appear in a similar set of documents, but these keywords would very likely have a similar semantic meaning (e.g., medical terms will appear in similar documents). Therefore, one might argue that, in this case, guessing that a keyword belongs to a group of words with similar semantic meaning can already be a privacy violation.



(a) Enron dataset

(b) Lucene dataset

Figure 7: Accuracy of `sap` against SEAL defense for different values of multiplicative volume padding x (50 weeks, $\bar{\eta} = 5$ queries/week).

Hiding the Search Pattern with Fresh Randomness. The schemes we have considered in this work leak the search pattern because the same keyword always produces the same access pattern. A scheme that generates access patterns with fresh randomness could prevent this from happening. A possible solution for this would be using an ORAM (e.g., TwoRAM [11]) scheme to hide which documents are retrieved from the dataset, and randomize the volume padding independently in every query. The problem with this solution is that ORAM-based SSE schemes incur considerable communication costs.

Even if the client was able to generate independent random access patterns for each query, the adversary could try to cluster similar access patterns together (two queries for the same keyword might still produce statistically similar access patterns since they aim to return the same set of documents). This clustering algorithm would be used to tag the observed queries. This tagging process would have some errors, that in the end would lower the accuracy of the query identification attack. It is however unclear how to build an *efficient* SSE scheme with independent access pattern obfuscation for each query such that access patterns are hard to cluster by keyword.

Hiding the Query Frequencies with Dummy Queries. A third alternative that has not been thoroughly explored in the literature is, instead of hiding the search patterns, obfuscating the query frequencies themselves by performing *dummy queries*. There are two immediate problems with this approach: first, it is not clear how to choose *when* to generate dummy queries without leaking whether the query is real or not through timing information. Generating a deterministic set of dummy queries for each real query [22] reveals more information and is less efficient than just merging these keywords in the search index (the first solution we mentioned in this section). A possible solution to this problem could come from anonymous communication technologies that already use traffic analysis-resistant dummy strategies (e.g., the Poisson cover traffic in Loopix [25]). Another problem of hiding query frequencies with dummy queries is *how* to choose the keywords of the dummy queries without requiring the client to store the set of all possible keywords in its local storage.

Even if the client implemented a dummy generation strategy, the adversary would know the particulars of this method and could adapt the attack accordingly, making corrections to the observed frequencies and limiting the effectiveness of the defense. Therefore, hiding the true frequency of queries with reasonable bandwidth overhead might be challenging.

8 Conclusions

In this work, we propose a query recovery attack against privacy-preserving Symmetric Searchable Encryption (SSE) schemes that support point queries. We derive this attack by setting up a maximum likelihood estimation problem and computing its solution by solving an unbalanced assignment problem. Unlike previous attacks, our proposal combines both volume information, computed from the access pattern leakage, and frequency information, obtained from the search pattern leakage. We show that, even in cases where taking this information separately does not pose a threat to the client's privacy, the combined information allows surprisingly high query recovery rates.

We consider different privacy-preserving SSE schemes that hide access pattern information and show how to adapt our attack against them. Our evaluation confirms that two of these defenses fail at providing a significant level of protection even when they are configured for high privacy regimes. The third defense is effective at hiding the query volume information, but even a small amount of frequency data (250 possibly repeated queries from the client, when there are 1 000 possible

keywords) can provide non-trivial query recovery rates (23%).

We hope that our work inspires researchers to find solutions that not only hide the access pattern leakage but also reduce the search pattern leakage, which we believe is paramount towards achieving effective privacy-preserving SSE schemes.

Acknowledgments

We gratefully acknowledge the support of NSERC for grants RGPIN-05849, CRDPJ-531191, IRC-537591 and the Royal Bank of Canada for funding this research. This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

Availability

Our code is available at https://github.com/simon-oya/USENIX21-sap-code.

References

[1] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. The tao of inference in privacy-protected databases. *Proceedings of the VLDB Endowment*, 11(11):1715–1728, 2018.

[2] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *Network and Distributed System Security Symposium (NDSS)*, page TBD, 2020.

[3] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 668–679, 2015.

[4] Guoxing Chen, Ten-Hwang Lai, Michael K Reiter, and Yinqian Zhang. Differentially private access patterns for searchable symmetric encryption. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 810–818. IEEE, 2018.

[5] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[6] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

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

[8] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[9] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[10] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K Cunningham. Sok: Cryptographically protected database search. In *IEEE Symposium on Security and Privacy (SP)*, pages 172–191. IEEE, 2017.

[11] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. Tworam: efficient oblivious ram in two rounds with applications to searchable encryption. In *Annual International Cryptology Conference*, pages 563–592. Springer, 2016.

[12] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[13] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *IEEE Symposium on Security and Privacy (SP)*, pages 1067–1083. IEEE, 2019.

[14] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. Encrypted databases: New volume attacks against range queries. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 361–378, 2019.

[15] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Network and Distributed System Security Symposium (NDSS)*, volume 20, page 12, 2012.

[16] Seny Kamara and Tarik Moataz. Encrypted multi-maps with computationally-secure leakage. *IACR Cryptology ePrint Archive*, 2018:978, 2018.

[17] Seny Kamara, Tarik Moataz, and Olya Ohrimenko. Structured encryption and leakage suppression. In *Annual International Cryptology Conference*, pages 339–370. Springer, 2018.

[18] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'neill. Generic attacks on secure outsourced databases. In *ACM SIGSAC Conference on Computer*

and Communications Security (CCS), pages 1329–1340, 2016.

[19] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. *IEEE Symposium on Security and Privacy (SP)*, pages 599–616, 2020.

[20] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[21] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *IEEE Symposium on Security and Privacy (SP)*, pages 297–314. IEEE, 2018.

[22] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-An Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 265:176–188, 2014.

[23] Omkant Pandey and Yannis Rouselakis. Property preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 375–391. Springer, 2012.

[24] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 79–93, 2019.

[25] Ania M Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. In *USENIX Security Symposium*, pages 1199–1216, 2017.

[26] David Pouliot and Charles V Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1341–1352, 2016.

[27] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. A path following algorithm for graph matching. In *International Conference on Image and Signal Processing*, pages 329–337. Springer, 2008.

[28] 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 Symposium*, pages 707–720, 2016.

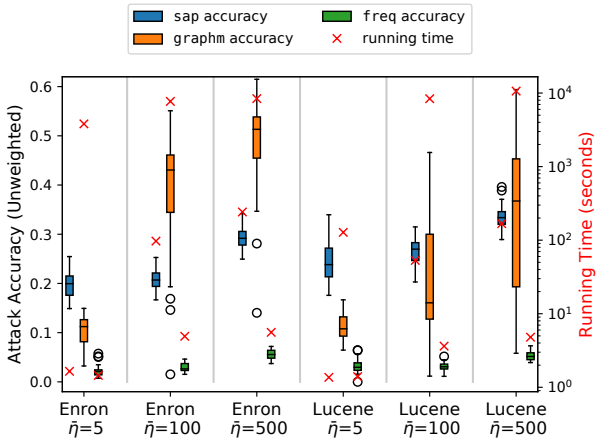


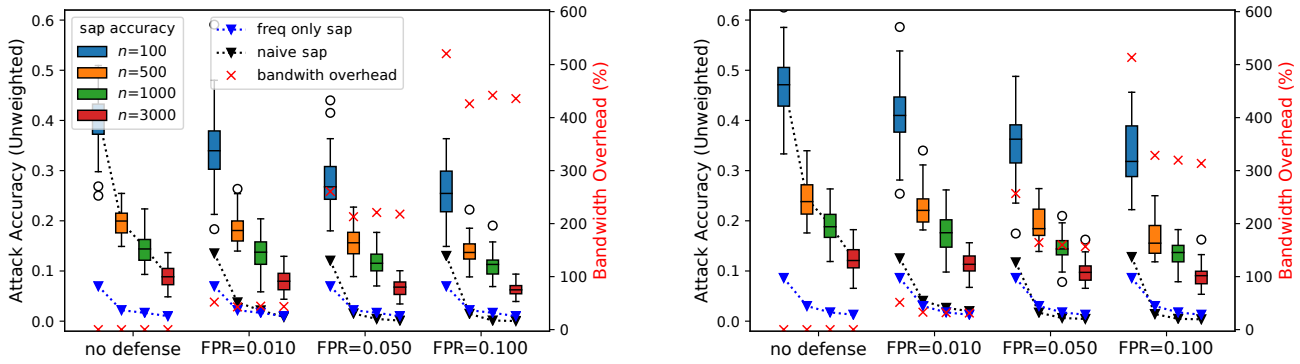
Figure 8: Unweighted recovery accuracy (boxes) and running time (×) of attacks in different datasets with $\bar{\eta}$ queries per week (50 weeks), with $n = 500$ keywords.

A Results as Percentage of Distinct Keywords Recovered

In Section 6, we measure the attack accuracy as the percentage of queries correctly recovered. In this section, for completeness, we report the accuracy of our experiments as the percentage of *unique keywords* the attack correctly identifies. We call this the *unweighted accuracy*, since it is not weighted by the number of times the client queries for each keyword.

Figure 8 shows the comparison between attacks in terms of unweighted accuracy (regular accuracy in Figure 4 — note the y-axes are different). Both `sap` and `freq` achieve lower unweighted accuracy than regular (weighted) accuracy, since they are more likely to correctly recover queries corresponding to frequently queried keywords. The unweighted accuracy of `graphm` is only slightly smaller than its regular accuracy; we conjecture this is because those keywords that are more popular in the dataset, and thus are easier to recover with co-occurrence information, are queried more often than unpopular keywords. Even though `graphm` performs on average better than `sap` when the adversary observes a large number of queries, we note that `graphm` is still 1) computationally unfeasible for large keyword universe sizes, 2) performs worse than `sap` both in weighted and unweighted accuracies when the client performs few queries per week, and 3) completely fails against defenses such as PPYY [24] and SEAL [7].

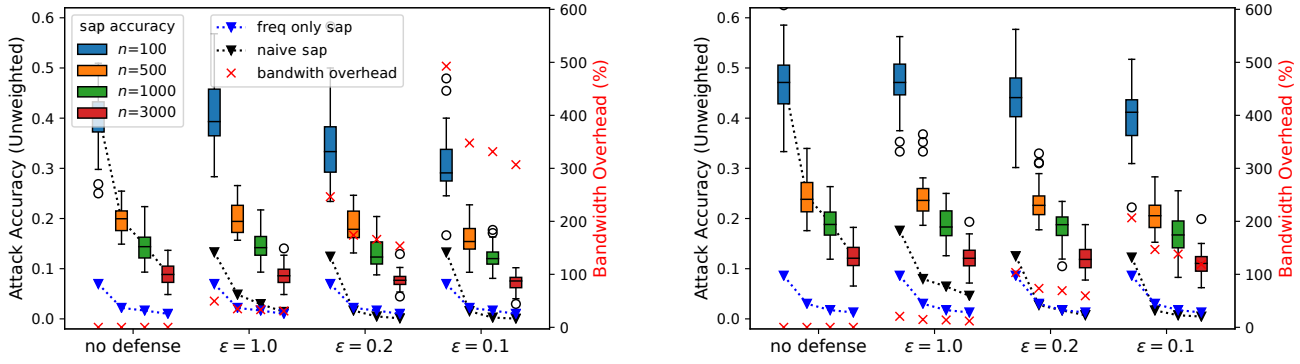
Figures 9 to 11 show the performance of `sap` in terms of the unweighted accuracy versus the three defenses we consider in the paper (the results for the regular accuracy are in Figures 5 to 7). Although the average number of unique keywords recovered by the attack is smaller than the average number of queries recovered, the results are qualitatively the same.



(a) Enron dataset

(b) Lucene dataset

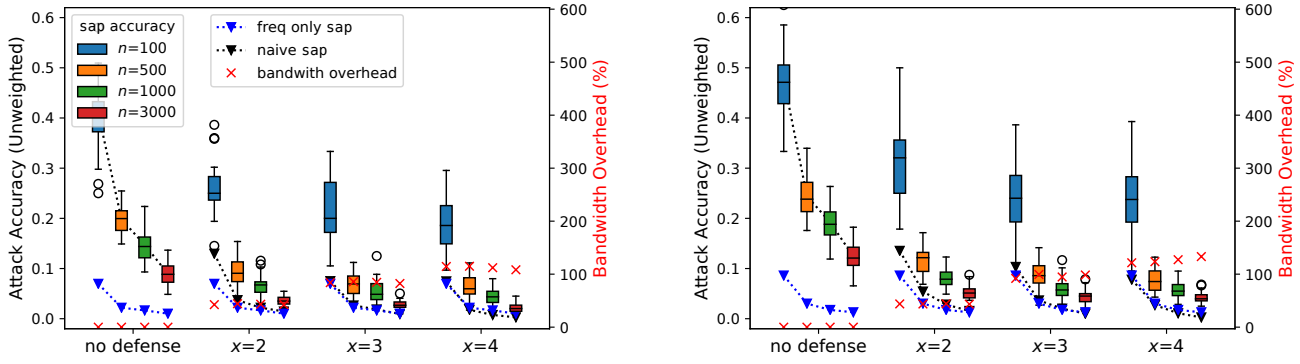
Figure 9: Unweighted accuracy of `sap` against CLRZ defense configured with $\texttt{TPR} = 0.999$ and varying $\texttt{FPR}$ (50 weeks, $\bar{\eta} = 5$ queries/week).



(a) Enron dataset

(b) Lucene dataset

Figure 10: Unweighted accuracy of `sap` against PPYY defense with different privacy values ε (50 weeks, $\bar{\eta} = 5$ queries/week).



(a) Enron dataset

(b) Lucene dataset

Figure 11: Unweighted accuracy of `sap` against SEAL defense for different values of multiplicative volume padding x (50 weeks, $\bar{\eta} = 5$ queries/week).