No Such Thing As A Small Leak: Leakage-Abuse Attacks Against Symmetric Searchable Encryption

Alexandre Anzala-Yamajako[1], Olivier Bernard[2], Matthieu Giraud[3], and Pascal Lafourcade[3]

[1] European Patent Office, Munich, Germany
ayamajakoanzala@epo.org
[2] Thales Communication and Security, CHiffre, Gennevilliers, France
olivier.bernard2@thalesgroup.com
[3] Université Clermont Auvergne, LIMOS, Aubière, France,
firstname.lastname@uca.fr

Abstract. *Symmetric Searchable Encryption* (SSE) schemes enable clients to securely outsource their data while maintaining the ability to perform keywords search over it. The security of these schemes is based on an explicit leakage profile. [16], has initiated the investigation into how much information could be deduced in practice from this leakage. In this paper, after recalling the leakage hierarchy introduced in 2015 by Cash et al. and the passive attacks of [16] on SSE schemes. We demonstrate the effectiveness of these attacks on a wider set of real-world datasets than previously shown. On the other hand, we show that the attacks are inefficient against some types of datasets. Finally, we used what we learned from the unsuccessful datasets to give insight into future countermeasures.

Keywords: symmetric searchable encryption, leakage, passive attacks

1 Introduction

The importance of digital data in everyday life is no longer in doubt. Their handling must be done with care leading to create and manage backups. To have access to those backups from anywhere and from different devices, outsourcing this digital data to a cloud provider is an enticing solution. The character of this data can be sensitive and/or confidential since some of this data are legal documents, banking and medical, industrial patents or emails. However, we host this digital data in all sorts of untrusted environments, potentially with unknown server administrators, OS and hypervisors. The trivial solution for a client is to use symmetric encryption. Indeed, since data is encrypted with a secret key unknown by the cloud provider, data confidentiality is no longer a problem. However, symmetric encryption prevents any server-side processing of the client data as is the norm on plaintext data. In particular, the processing of search queries by the server sent by the client is no longer possible. Indeed,

if the client gives a keyword to the server, the latter cannot retrieves the documents containing that keyword since documents are encrypted. On the other hand, fully homomorphic encryption [15] allows the server to execute directly in the encrypted domain search operation needed to answer search queries. Unfortunately, such an approach would solve our problem only from a theoretical point of view because making a fully homomorphic encryption scheme work in practice remains an open question (as noted e.g., in [15]). *Symmetric Searchable Encryption* (SSE) schemes introduced by Song et al. in [26] aim at retaining this search capability on encrypted data. SSE scheme is a protocol executed between a client and a server. We consider a client owning a sensitive/confidential set of plaintext documents stored in a *DataBase* denoted DB. We assume that this client has limited computational power and storage capacity. On other hand, we consider a server having a large storage space and high processing power. This server is *honest-but-curious* [17], i.e., it is not trusted by the client except for executing correctly the search protocol. First, an SSE scheme creates, from DB, metadata that is protected in an *Encrypted DataBase* (EDB). Then the client outsources EDB on the server along with the encryptions of the documents. After that, the client can send a search token generated from a keyword and her symmetric secret key. With this search token, the server can find the encrypted documents matching the query with the help of EDB. Finally, the corresponding encrypted documents are sent back to the client for decryption. The basic functionality of an SSE scheme is to retrieve the encrypted documents matching one single keyword query. However, there exist SSE schemes allowing the client to add new encrypted documents to the encrypted database while retaining the search capability; these schemes are called *dynamic* SSE [10, 21]. Others SSE schemes focus on expanding the expressiveness of the search queries such as Boolean [11] and sub-string search queries [14].

By its nature, a SSE scheme reveals to an observer the *search* pattern, indicating which other queries were also for the considered keyword and representing the fact that SSE scheme sends the same search token when a search is repeated. It also reveals the *access* pattern, showing the identifiers of documents concerned by the search token. This kind of information leaked by a given SSE scheme to the server is formalized by a *leakage function* [12, 21]. The security of a SSE scheme is based on this function proving that the scheme does not leak more information than described in the leakage function. However, this leaked information can be used by an honest-but-curious server executing dutifully the scheme to to deduce information on the stored documents. For instance, some *inference attacks* [20, 9, 25] use these search and access patterns to reconstruct DB. In this paper, we show that it is possible to reconstruct DB without using the search and access pattern leading to passive attacks. We focus on the information revealed by the encrypted database EDB regardless exchanges between the client and the server. We argue that this model is realistic since it allows for the server itself (or any entity having access to the encrypted database, as the server administrator or the OS) to be malicious by manipulating the leaked information. Based on deployed SSE schemes, Cash et al. [9] define four *leakage*

profiles L4, L3, L2 and L1 depending on leaked information by the schemes. The L4 profile represents SSE schemes that are the most leaky while the L1 profile represents the least leaky schemes. Commercially available SSE solutions such as CipherCloud[4] and Skyhigh Networks[5] are L4 schemes, while Bitglass[6] (resp. ShadowCrypt[7]) is a L3 scheme (resp. L2 scheme). On the contrary, most SSE schemes proposed in academic research have L1 profile. The advantage of the L4-, L3- and L2-SSE schemes is that they can be set up as a proxy or as extensions in client-side. Indeed, no modification on server-side is necessary. Although these solutions are tempting, it is important to study the practical impacts of these information leaks on the document knowledge of the server. In this paper, we study the impact of a passive attacker (as the server storing EDB).

Our contributions. We design passive attacks against SSE schemes of L4, L3 and L2 leakage profiles in order to reconstruct as much as possible the database DB. Our attacks do not rely on observing search queries, neither on the access pattern. We assume that the server, storing the encrypted documents and the encrypted database EDB, only knows a small sample of plaintexts. Using this known sample and the leaked information by the scheme on the encrypted database EDB, we start by finding the identifiers in the encrypted database associated to the plain documents of the sample. This first step leads to correspondences between plain documents and their representation in the encrypted database. Then, the adversary considers each correspondence and tries to deduce values of plain keywords in the encrypted database. Since L4-, L3- and L2-SSE schemes replace each plain keywords by the same value, these keywords-values associations are used to recover other documents that are not in the sample known by the adversary. Our attack on L4 schemes uses repetitions and order of keywords in each document, our attack on L3 schemes uses order of shared keywords between documents while our attack on L2 schemes uses only information on shared keywords between documents. We demonstrate that the efficiency and practicality of our three passive attacks depend on the nature of the data set. In fact, when we consider data sets such as mailing-list, emails, or books, we show that an adversary knowing only 1% of plain documents is able to reconstruct between 65 % and 90% of the protected data at 80%. On the contrary, when we consider data sets constitued of server logs or of movies descriptions, our attacks are practically inefficient. In fact, if the adversary knows only 1% of plain documents, she is able to reconstruct at most less than 5% of the protected data at 80% for SSE schemes of L4 and L3 leakage profile, while she is not able to do any associations when we consider SSE scheme of L2 leakage profile.

Related Work. Considering an *active* adversary that is able to add chosen documents in the database of the client, Cash et al. [9] present a partial document

[4] https://ciphercloud.com/technologies/encryption/
[5] https://skyhighnetworks.com/product/salesforce-security/
[6] https://bitglass.com/salesforce-security
[7] https://shadowcrypt-release.weebly.com/

recovery attack on L3- and L2-SSE schemes using frequency distribution of keywords hashes. Moreover, Zhang et al. [28] consider an adversary that has the extra ability to issue selected queries and mount a query recovery attack that works on any dynamic SSE scheme. These active attacks are very efficient. They reveal associations between keywords and search tokens with only few injected files. Contrary to these attacks, we consider only a passive adversary who is not able to plant document in the database or to issue particular queries.

An other family of attacks, based on the search pattern and the access pattern have been also proposed and are called *inference attacks*. The first one is the *IKK Attack* and has been proposed by Islam et al. [20]. The goal of their attack is to associate search tokens sent by the client to actual keywords. In order to do that, they exploit the data access pattern revealed by client queries and assume that the adversary has access to a co-occurrence matrix. Each element of the co-occurrence matrix corresponds to the probability that two keywords appear in a randomly chosen document. However, Cash et al. [9] stress that this matrix needs to be so precise for the attack to succeed. This precision legitimates the assumption that the adversary has access to the number of documents in which every keyword appears. With this strong extra knowledge, authors mount a more effective attack named the *Count Attack* [9]. These two attacks target SSE schemes of L1 leakage profile. However, the strength of their assumptions questions their practicality. On the contrary, our attacks do not rely on observing client queries but only assume that the adversary knows the encrypted database, which totally natural when we consider the server as the adversary. In a different way, Abdelraheem et al. [4] proposed an inference attack when a client protect a relational database with a SSE scheme. Exploiting the structural properties of relational databases, the authors show that record-injection attacks mounted on relational databases have worse consequences than their file-injection counterparts on unstructured databases as data sets that we consider.

Cash et al. [9] also proposed a passive attack concerning L3-SSE schemes to partially recover documents when the adversary knows plaintext-ciphertext pairs. Unlike them, our attacks suppose that we do not have any plaintext-ciphertext pairs initially. An other inference attack proposed by Pouliot and Wright in [25] and called *Shadow Nemesis Attack* uses a training data set in order to build a co-occurrence matrix as in [9]. This co-occurrence matrix is then reduced to the problem of matching search tokens to keywords to the combinatorial optimization problem of weighted graph matching. The *Shadow Nemesis Attack* can be performed on L2-SSE schemes as our attacks. While our attacks use a sample of plain documents of the original database DB, their attack can use a training data set instead of a partial knowledge of the original data set in addition of the encrypted database.

Wang et al. [27] also present inference attacks on searchable encryption. As we do, they consider passive attacks where an adversary knows a subset of documents. Authors propose an other leakage model where the one-to-one mapping between keywords and tokens change to a one-to-many mapping (as used in Mimesis aegis [22]), increasing the difficulty of statistical analysis. They algo-

rithms and attacks follow the same idea as our paper [16] then without any surprise they results are similar to our results. Moreover they claim that our attacks cannot be applied to index-based SSE schemes, which is not true at all.

Our article is an extended version of [16]. The main difference is that we selected nine different datasets samples and evaluate the impact of our attacks showing that the efficiency of our passive attacks depends on the nature of the data sets. Indeed, we show that our attacks are very efficient against database containing books, emails or mailing-list. On the contrary, data sets such as server logs or movies descriptions are resistant to such passive attacks. Moreover, we give statistic measures the different data sets, including the number of documents sharing the same length in $\mathsf{L4}$ point of view, the number of documents sharing the same length in the $\mathsf{L3}$ point of view, and the occurrence of each keywords for each data set. We experiment our attacks on these samples as detailed in Section 5, and we also add results of our attacks for three datasets in full among the selected samples, namely *Commons*, *Hadoop*, and *Lucene*.

Outline. We start by providing background on SSE schemes and their security recalling notations in Section 2. Then we recall in Section 3 the leakage hierarchy of SSE schemes presented in [9]. We describe our new passive attacks in Section 4 and we demonstrate their effectiveness in Section 5 before to conclude in Section 6.

2 Symmetric Searchable Encryption

The notations and definitions provided in this section are shorter version of the ones presented in [16]. They have been included here to ease the reader understanding of SSE schemes but the reader is refered to [16] for a more complete presentation.

2.1 Notations

Sequences, lists and sets. We define a sequence of elements as an ordered set where repetitions are allowed. A list is therefore an ordered set where all elements are distinct, i.e., there is no repetition. Finally, a set is defined as an unordered bunch of distinct elements. We denote sequences by parenthesis $(\dots)$, lists by square brackets $[\dots]$ and sets by braces $\{\dots\}$. Let E be a set (resp. list or sequence), then we denote the number of elements in E by $\#E$.

Documents and keywords. Let $\mathsf{DB} = \{d_1, \dots, d_n\}$ be a set of n documents. We recall that DB is called the *data set*. Each document of DB is composed of keyword belonging to a dictionary $\mathsf{W} = \{w_1, \dots, w_m\}$ made of m keywords. Moreover, each document $d_i \in \mathsf{DB}$ is a sequence of length ℓ_i, in other terms $d_i = (w_{i_1}, \dots, w_{i_{\ell_i}}) \in \mathsf{W}^{\ell_i}$. We denote by W_i the set of distinct keywords of the document d_i, i.e., $\mathsf{W}_i = \{[d_i]\}$.

When the same objects are considered on server-side, we describe them by introducing the star superscript. Hence, we denote the set of search tokens associated to the keywords of W by $\mathsf{W}^* = \{w_1^*, \ldots, w_m^*\}$. In the same way, we denote the set of ciphertexts of DB by $\mathsf{DB}^* = \{d_1^*, \ldots, d_n^*\}$ where d_i^* is the encryption of d_i. Finally, we denote the set of tokens associated to d_i^* by W_i^*. Since the association between between d_i and d_i^* is not known by the server a priori, we use an identifier, denoted id_i, to uniquely represent d_i^*. Moreover, a datastructure EDB is also provided. It contains protected metadata that allows the server to answer search queries sent by the client.

Considering a keyword w, we denote by $\mathsf{DB}(w)$ the list of all the indices i such that $d_i \in \mathsf{DB}$ contains the keyword w. We denote by N, the number of pairs (d, w) where $d \in \mathsf{DB}$ and $w \in d$, i.e. $N = \#\{(d, w) \mid d \in \mathsf{DB}, w \in d\}$. We remark that N corresponds to a lower bound on the size of EDB. Indeed, N can always be computed by the server from EDB. Server-side, the list of the identifiers of all the documents $d_i^* \in \mathsf{DB}^*$ associated to the search token w^* is denoted $\mathsf{EDB}(w^*)$. We stress that this information is not accessible directly from w^* and DB^*, we need the extra protected metadata structure EDB.

2.2 SSE Schemes

The basic definition of a symmetric encryption scheme is given with an algorithm for setup and another for search.

First, the client generates two datastructures denoted DB^* and EDB. As defined above, DB^* is composed of ciphertexts of DB, and EDB contains protected metadata associated to DB. Then the client outsources these two datastructures to the server. Assuming the client wants to search for a specific keyword w, she computes with the help of her secret key the search token w^* associated to w, and sends it to the server. When the server receives the search token w^* from the client, the server uses EDB to return the identifiers of all encrypted documents matching the client's search. From this list of identifiers, the client retrieves the associated ciphertexts and decrypt them in order to obtain the plaintext documents. We stress that the server should not be able to learn anything about the client's query or the returned documents during the process. For further details, see Definition 1.

Definition 1 (Static SSE scheme [12]). *Given a symmetric encryption scheme* $(\mathcal{E}.(\cdot), \mathcal{D}.(\cdot))$ *we define a* static SSE scheme *of security parameter* λ *as a quartet of polynomial-time algorithms* $\Pi = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{SearchClient}, \mathsf{SearchServer})$ *by:*

$(K, k) \leftarrow \mathsf{Gen}(1^\lambda)$ *is a probabilistic key generation algorithm that is run by the client. It takes as input a security parameter* λ*, and outputs two symmetric secret keys* K *and* k *which are both kept securely by the client.*

$(\mathsf{EDB}, \mathsf{DB}^*) \leftarrow \mathsf{Setup}(K, k, \mathsf{DB}, \mathcal{E}.(\cdot))$ *is an algorithm that is run by the client to set the scheme up. It takes as input secret keys* K *and* k*, the database* DB *and the algorithm* $\mathcal{E}.(\cdot)$*, and outputs both the protected metadata* EDB *and the encrypted documents* $\mathsf{DB}^* = (\mathcal{E}_k(d_1), \ldots, \mathcal{E}_k(d_n))$.

$w^* \leftarrow \mathsf{SearchClient}(K, w)$ *is a deterministic algorithm that is run by the client to send a query to the server. It takes as input the secret key* K *and a keyword queried* $w \in \mathsf{W}$*, and outputs the search token* $w^* \in \mathsf{W}^*$ *associated with* w*. Finally* w^* *is sent to the server.*

$\mathsf{EDB}(w^*) \leftarrow \mathsf{SearchServer}(\mathsf{EDB}, w^*)$ *is a deterministic algorithm that is run by the server to answer a client-query. It takes as input the protected metadata* EDB *and the client-generated search token* w^* *and outputs* $\mathsf{EDB}(w^*)$*: the identifiers of the encrypted documents containing keyword* w*. This list is sent back to the client.*

2.3 Security of SSE Schemes

The notion of a *leakage function* of a SSE scheme has been introduced by Curtmola et al. [12]. The leakage function, denoted $\mathcal{L}$, of a SSE scheme is the set of information revealed by the SSE scheme to the server. It formalizes the information leaked to the server by both the encrypted database EDB and by the client queries.

An SSE scheme is said to be $\mathcal{L}$-secure if no information other than what is described by the leakage function is leaked by the SSE scheme to the server. More precisely, the $\mathcal{L}$-security of a SSE scheme proves that any polynomial-time adversary making a sequence of queries, constitued of keywords of W, can successfully tell with only negligible probability whether the protocol is honestly executed or simulated from the leakage function $\mathcal{L}$.

Although the leakage function described a bound of leaked information by the SSE scheme on the stored documents, it does not tell us what this leakage implies *in practice* on the knowledge of the protected data. The objective of [16] was to present algorithms that gave more insight into the practical impact of such leakage for the different leakage profiles presented above. In this work we focus on the difference of effectiveness of those algorithms when faced with different document sets and investigate the reasons for those differences.

3 A Leakage Hierarchy

Similarly as in Section 2, this section is a shorter version of the content presented in [16]. It has been included here for the article to be self-contained but the reader is referred to [16] for a more complete presentation.

$\mathsf{L4}$ *Leakage Profile.* Without taking in consideration the meaning of words, a document is described by the number of its words, their order and their occurrence. Moreover, a document can be described by words shared with other documents. $\mathsf{L4}$-SSE schemes reveal this information, so nothing is lost about the plaintext non-semantic structure. So, a SSE scheme of leakage function $\mathcal{L}$ is of class $\mathsf{L4}$ if and only if:

$$\mathcal{L}(\mathsf{EDB}) = \left\{(w^*_{i_1}, \ldots, w^*_{i_{\ell_i}})\right\}_{1 \leqslant i \leqslant n}.$$

L3 *Leakage Profile.* For keyword search purposes, it is not necessary to know the occurrence count of each keyword. L3-SSE schemes of leakage function $\mathcal{L}$ do not reveal this, i.e.:

$$\mathcal{L}(\mathsf{EDB}) = \left\{\mathsf{L3}_{\mathsf{EDB}}(\mathsf{id}_i)\right\}_{1\leqslant i\leqslant n},$$

where $\mathsf{L3}_{\mathsf{EDB}}(\mathsf{id}_i) = \left[(w^*_{i_1}, \ldots, w^*_{i_{\ell_i}})\right]$.

L2 *Leakage Profile.* L2-SSE schemes only reveal the set of tokens of a document. The server can however still determine which documents contain a given token. A SSE scheme of leakage function $\mathcal{L}$ is of class L2 if and only if:

$$\mathcal{L}(\mathsf{EDB}) = \left\{W^*_i\right\}_{1\leqslant i\leqslant n}.$$

L1 *Leakage Profile.* With no initial search, L1-SSE schemes leak the least possible amount of information, i.e. the number N of document/keyword pairs of the dataset:

$$\mathcal{L}(\mathsf{EDB}) = \{N\}.$$

4 Partial Plaintext Recovery Attacks

The attacks presented in [16] are briefly recalled here. As previously pointed out, a more complete presentation can be found in [16]. We start by recalling the intuition behind the attacks as well some facts regarding their applicability.

These are passive attacks which aim at recovering plaintext information from the sole knowledge of EDB. The only assumption made here is that we know a small sample $\mathcal{S}$ chosen randomly among the set of plaintext documents.

The attacks proceed in two steps. In the first step, each plaintext of $\mathcal{S}$ is associated to its protected information in EDB. This step is performed using statistical properties that can be computed independently from the plaintexts themselves or from the associated leakage given in EDB. The efficiency of the attack heavily depends on the statistic capacity to give unique results over the dataset. In the second step, the keywords of the plaintexts are paired with their tokens. The matching keywords and tokens obtained can be spread back into EDB thus recovering content of the encrypted documents. [16] presented very positive results and the application of these same attacks to different datasets allows to gain more insights into their computational complexity as well as intuition for potential countermeasures. Those results are presented in Section 5.

4.1 Mask Attack on L4-SSE Schemes

We introduce the *mask* of a document d_i (resp. id_i), denoted by $\mathsf{mask}(d_i)$ (resp. $\mathsf{mask}(\mathsf{id}_i)$), as the sequence where all keywords (resp. tokens) are replaced by their position of first appearance.

The idea of the attack is intuitive: for each plaintext $d \in \mathcal{S}$, the mask of d is computed; this mask is then compared with all masks of corresponding length computed from EDB. The attack is summarized in Algorithm 1.

Algorithm 1: Mask Attack.

Input: $\mathsf{EDB}, \mathcal{S} \subseteq \mathsf{DB}$
Output: Set of tokens $\mathsf{W}^*_{\mathsf{rec}} \subseteq \mathsf{W}^*$ associated to their keyword in W

foreach $d \in \mathcal{S}$ **do**
  $A_d = \{i \mid \ell_i = \#d \text{ and } \mathsf{mask}(\mathsf{id}_i) = \mathsf{mask}(d)\}$;
end
return $\mathsf{W}^*_{\mathsf{rec}} = \{\mathsf{W}^* \mid \#A_{d_i} = 1\}$

4.2 Co-Mask Attack on L3-SSE Schemes

Here, we introduce the *co-resulting mask* of a pair (d_1, d_2) of documents, denoted by $\mathsf{comask}(d_1, d_2)$. Intuitively, it can be viewed as the mask of positions of shared keywords in the other document. More formally, if $\mathsf{Pos}(w, d)$ is the position of keyword w in document $[d]$, we define:

$$\mathsf{comask}(d_1, d_2) = \Big(\big(\mathsf{Pos}(d_1[i], d_2)\big)_{1 \le i \le \#\mathsf{W}_1},\ \big(\mathsf{Pos}(d_2[i], d_1)\big)_{1 \le i \le \#\mathsf{W}_2} \Big) .$$

The co-resulting mask can also be computed directly from every pair of encrypted identifiers so by abuse of notation we denote it denoted by $\mathsf{comask}(\mathsf{id}_1, \mathsf{id}_2)$.

In practice, instead of randomly searching for matching pairs we iteratively construct a set A_t containing all t-tuples of identifiers such that the co-resulting masks of all pairs in the t-uple match the co-resulting masks of the corresponding pairs in $(d_1, \ldots, d_t) \subseteq \mathcal{S}$. More formally:

$$A_t = \Big\{ \big(\mathsf{id}_{i_1}, \ldots, \mathsf{id}_{i_t}\big) \mid \forall s, u \le t,\ \mathsf{comask}(\mathsf{id}_{i_s}, \mathsf{id}_{i_u}) = \mathsf{comask}(d_s, d_u) \Big\} .$$

The Co-Mask attack is summarized in Algorithm 2.

4.3 PowerSet Attack on L2-SSE Schemes

The loss of keyword-order information under L2 leakage presents us with two new challenges: first, the co-resulting mask cannot be computed and second, even if a document is correctly associated to its identifier, finding the correct association between each keyword and its token is still to be done. The PowerSet Attack addresses both issues.

Associating Documents and Identifiers. We introduce the *power set of order h* of a list of t documents, denoted by $\mathsf{PowerSet}^t_h(d_1, \ldots, d_t)$, and defined as the sequence of the $\binom{t}{h}$ cardinals of all possible intersections of h elements of the t-uple.

More formally:

$$\mathsf{PowerSet}^t_h(d_1, \ldots, d_t) = \left(\# \bigcap_{1 \le j \le h} d_{i_j} \right)_{1 \le i_1 < \cdots < i_h \le t} .$$

Algorithm 2: Co-Mask Attack.

Input: EDB, $\mathcal{S} = \big(d_1, \ldots, d_{\#\mathcal{S}}\big) \subseteq \mathsf{DB}$
Output: Set of tokens $\mathsf{W}^*_{\mathsf{rec}} \subseteq \mathsf{W}^*$ associated to their keyword in W

`/* Consider the first pair of documents */`
$A_2 = \left\{ \big(\mathsf{id}_{i_1}, \mathsf{id}_{i_2}\big) \;\middle|\; \begin{array}{l} \#\mathsf{id}_{i_1} = \#[d_1],\ \#\mathsf{id}_{i_2} = \#[d_2] \\ \text{and } \mathsf{comask}(\mathsf{id}_{i_1}, \mathsf{id}_{i_2}) = \mathsf{comask}(d_1, d_2) \end{array} \right\}$;
`/* Construct` A_t `from` A_{t-1} `using` d_t `*/`
for $t = 3$ **to** $\#\mathcal{S}$ **do**
  $A_t = A_{t-1} \times \big\{\mathsf{id} \mid \#\mathsf{id} = \#[d_t]\big\}$;
  `/*` A_t `will be reduced by considering all new pairs` (d_j, d_t) `*/`
  foreach $j < t$ **do**
    $C_{j,t} = \left\{ \big(\mathsf{id}_{i_j}, \mathsf{id}_{i_t}\big) \;\middle|\; \begin{array}{l} \mathsf{id}_{i_j} \in A_t[j],\ \mathsf{id}_t \in A_t[t] \\ \text{and } \mathsf{comask}(\mathsf{id}_{i_j}, \mathsf{id}_{i_t}) = \mathsf{comask}(d_j, d_t) \end{array} \right\}$;
    $A_t = \big\{a \in A_t \mid \big(a[j], a[t]\big) \in C_{j,t}\big\}$; `/* Keep consistent` t`-tuples */`
    if $\#A_t = 1$ **then** break;
  end
end
return $\mathsf{W}^*_{\mathsf{rec}} = \big\{\mathsf{W}^*_t \mid \#A_{\#\mathcal{S}}[t] = 1\big\}$

The superscript will be omitted when it is clear from the context and since this sequence can also be computed directly from the list of encrypted identifiers so by abuse of notation we denote it denoted by $\mathsf{PowerSet}_h\big(\mathsf{id}_1, \ldots, \mathsf{id}_t\big)$.

We again iteratively construct a set A_t containing all t-tuples of identifiers such that all power sets of order less than t correspond to the power sets of the corresponding documents in $(d_1, \ldots, d_t) \in \mathcal{S}$. When t reaches $\#\mathcal{S}$, all information on $\mathcal{S}$ has been processed and singleton components of $A_{\#\mathcal{S}}$ give a correct association.

Let $A_t^{(h)}$ be the set of compatible t-tuples with all power sets of order up to h:

$$A_t^{(h)} = \left\{ \big(\mathsf{id}_{i_1}, \ldots, \mathsf{id}_{i_t}\big) \mid \forall s \leq h, \quad \begin{array}{c} \mathsf{PowerSet}_s(d_1, \ldots, d_t) \\ = \mathsf{PowerSet}_s(\mathsf{id}_{i_1}, \ldots, \mathsf{id}_{i_t}) \end{array} \right\} .$$

The algorithm then computes the following decreasing sequence, using the procedure Reduce given in Algorithm 3 to go from $A_t^{(h)}$ to $A_t^{(h+1)}$:

$$A_{t-1} \times \big\{\mathsf{id} \mid \#\mathsf{id} = \#\{d_t\}\big\} = A_t^{(1)} \supseteq A_t^{(2)} \supseteq A_t^{(3)} \supseteq \cdots \supseteq A_t^{(t)} = A_t \ .$$

Algorithm 4 summarizes the first phase of the PowerSet Attack.

Associating Keywords and Tokens. As token order is not preserved under $\mathsf{L2}$ leakage, finding the correct keyword-token associations remains non-trivial.

Algorithm 3: Reduce procedure: computing $A_t^{(h+1)}$ from $A_t^{(h)}$.

Input: $\mathcal{S}_t = \big(d_1, \ldots, d_t\big)$, h-order candidates $A_t^{(h)}$
Output: Set of $(h+1)$-order candidates $A_t^{(h+1)}$

$B_t = A_t^{(h)}$;
`/* Consider each subset of` $(h+1)$ `elements containing` d_t `*/`
foreach $1 \leq j_1 < \cdots < j_h < t$ **do**
  $C_{j,t} = \left\{ \big((\mathsf{id}_{i_j}), \mathsf{id}_{i_t}\big) \;\middle|\; \begin{array}{l} \mathsf{id}_t \in B_t[t],\ (\mathsf{id}_{i_j}) \in B_t[j], \\ \text{and } \#\big(\mathsf{id}_{i_t} \cap (\mathsf{id}_{i_j})\big) = \#\big(d_t \cap (d_j)\big) \end{array} \right\}$;
  $B_t = \big\{ b \in B_t \;\mid\; \big((b[j]), b[t]\big) \in C_{j,t} \big\}$; `/* Keep consistent` t`-tuples */`
  if $\#B_t = 1$ **then** break;
end
return $A_t^{(h+1)} = B_t$

Algorithm 4: PowerSet Attack: documents-identifiers association.

Input: EDB, $\mathcal{S} = \big(d_1, \ldots, d_{\#\mathcal{S}}\big) \subseteq \mathsf{DB}$
Output: Set of documents $\mathcal{S}_0 \subseteq \mathcal{S}$ associated to their identifiers in EDB

`/* Consider the first pair of documents */`
$A_2 = \left\{ \big(\mathsf{id}_{i_1}, \mathsf{id}_{i_2}\big) \;\middle|\; \begin{array}{l} \#\mathsf{id}_{i_1} = \#\{d_1\},\ \#\mathsf{id}_{i_2} = \#\{d_2\} \\ \text{and } \mathsf{PowerSet}_2(\mathsf{id}_{i_1}, \mathsf{id}_{i_2}) = \mathsf{PowerSet}_2(d_1, d_2) \end{array} \right\}$;
`/* Construct` A_t `from` A_{t-1} `using` d_t `*/`
for $t = 3$ **to** $\#\mathcal{S}$ **do**
  $A_t^{(1)} = A_{t-1} \times \big\{ \mathsf{id} \;\mid\; \#\mathsf{id} = \#\{d_t\} \big\}$;
  `/* Consider intersections of increasing order` h `to reduce` A_t `*/`
  for $h = 2$ **to** t **do**
    $A_t^{(h)} = \mathsf{Reduce}\big(A_t^{(h-1)}\big)$;
    if $\#A_t^{(h)} = 1$ **then** set $A_t = A_t^{(h)}$ and break;
  end
end
return $\mathcal{S}_0 = \big\{ d_t \;\mid\; \#A_{\#\mathcal{S}}[t] = 1 \big\}$

To solve this problem, we construct the *inverted index* of $\mathcal{S}_0$, denoted by $\mathsf{inv}(\mathcal{S}_0)$, which associates the keywords $w \in \mathcal{S}_0$ and to the identifiers of the documents containing w. This inverted index is then ordered by decreasing number of identifiers to form the *ordered inverted index* $\mathsf{inv}_{\geq}(\mathcal{S}_0)$.

The keyword/token association process is given in Algorithm 5.

5 Experimental Results

Before to present the experimental results, we describe the datasets on which we ran the attacks of [16] reintroduced in Section 4.

Algorithm 5: PowerSet Attack: keywords-tokens association.

Input: EDB, set $\mathcal{S}_0 \subseteq \mathcal{S}$ of documents associated to their identifiers
Output: Set of tokens $\mathsf{W}^*_{\mathsf{rec}} \subseteq \mathsf{W}^*$ associated to their keyword in W

$\mathsf{W}^*_{\mathsf{ign}} \leftarrow \emptyset$; `/* Contains associated and indistinguishable tokens */`
Compute $\mathsf{inv}_{\geq}(\mathcal{S}_0)$;
foreach $w \in \mathsf{inv}_{\geq}(\mathcal{S}_0)$ *taken in decreasing order* **do**
  $A_w = \left(\bigcap \left\{\mathsf{W}^*_i \mid \mathsf{id}_i \in \mathsf{inv}_{\geq}(\mathcal{S}_0)[w]\right\}\right) \setminus \mathsf{W}^*_{\mathsf{ign}}$;
  $\mathsf{W}^*_{\mathsf{ign}} = \mathsf{W}^*_{\mathsf{ign}} \cup A_w$; `/* Associated (#`$A_w = 1$`) or indistinguishable */`
end
return $\mathsf{W}^*_{\mathsf{rec}} = \left\{A_w \mid \#A_w = 1\right\}$

5.1 Real-World Datasets

The attacks presented in Section 4 were ran on different real-world datasets to evaluate their practical efficiency. We choose 9 datasets and for each dataset, we select a random sample of 2,000 documents where we apply our attacks. We summarized the different samples in Figure 1. We recall that #DB is the number of documents in the dataset, #W is the number of distinct keywords in the dataset, and N denotes the number of pairs (d, w) where $d \in \mathsf{DB}$ and $w \in d$.

Dataset	Content	#DB	#W	N
Gutenberg	books	2,000	527,999	8,153,007
Commons	mailing list	2,000	40,924	264,554
Hadoop	mailing list	2,000	46,097	343,428
Lucene	mailing list	2,000	38,619	269,718
Subversion	mailing list	1,909	35,190	319,845
Enron	emails	2,000	25,998	184,202
Wikipedia	articles	2,000	60,379	335,620
IMDb	movie database	2,000	5,671	20,457
Nasa	server logs	2,000	2,280	17,971

Fig. 1. Details on selected samples of the datasets.

The first dataset sample is the *Project Gutenberg* dataset [18] containing full texts of public domain books. The next four datasets samples are mailing lists from the *Apache* foundation, namely *Apache Commons* [5], *Apache Hadoop* [6], *Apache Lucene* [7], and *Apache Subversion* [8]. The following dataset sample is the email dataset from the *Enron* corporation, available online [13]. Then, the *Wikipedia* dataset sample is formed of articles coming from the online encyclopedia Wikipedia[8]. The following dataset is *IMDb* [19], it is composed of movies descriptions as an alphanumeric unique identifier of the title, the type/format of the title (e.g. movie, short, video, etc.), the more popular title, the original

[8] https://en.wikipedia.org/

title, the release and the end years, the primary runtime of the title, and the genre. The final dataset is *Nasa* consisting of server logs of `http` requests to the NASA Kennedy Space Center web server in Florida [23]. They are composed of hostname, timestamp, request, HTTP reply code, and the number of bytes in the reply.

One email, one book, one message, one article, one movie description, or one log server is considered as one document. For each document, stopwords have been removed and keywords processed using the standard Porter stemming algorithm [24].

Those datasets were chosen for their variety as they are a representative sample of datasets that one might want to protect with SSE schemes. As we will see in the following the efficiency of the attacks varies wildly with the type of dataset considered.

5.2 Foreseeing the Cost of the Attacks

In [16] the computational cost of the attacks sketched by making some assumptions on the statistical properties of the considered dataset. Further experiments have shown that those assumptions, while holding true for the datasets considered in [16], could also be far from the empirical truth when looking at other types of datasets. Therefore, we take a different approach as the one chosen in [16]: instead of deriving formulas for the computational complexity of an attack based on assumptions about the dataset, we measure parameters of the dataset and derive from that the approximate difficulty of running the attack *on this particular dataset*. We believe that this approach is more useful to evaluate the practical security of a given SSE scheme as this can only be evaluated once faced with a dataset. The other advantage of this approach is that it provides insight towards possible counter-measures against the attacks presented above.

The governing parameter for the cost of the mask attack is the number of candidate masks that need to be computed for each plaintext document. This is exactly the number of encrypted documents having the same length as the target plaintext. Therefore, figure 2 shows the number of documents having the same length in the view of the $\mathsf{L4}$ leakage profile for our 9 datasets, i.e., we take in account the repetition of keywords in the documents. The y-axis of figure 2 represents the number of documents sharing a given length. From the figure we can predict that the mask attack will be extremely efficient on the dataset *Gutenberg* for which there is at most 3 encrypted candidate for each target plaintext. At the other end of the spectrum, all the documents of the *Nasa* dataset have length 10. This effectively means that each target plaintext requires to compute the mask of *all* encrypted documents. This property makes the attack extremely slow and effectively impractical. These results follow the intuition that a dataset of documents of arbitrary length will be far easier to attack that a dataset made of very formatted documents. This intuition is further confirmed by the results of the other 7 datasets: the non-formatted datasets *Commons*, *Hadoop*, *Lucene*, *Subversion*, *Enron*, and *Wikipedia* force the computation of less than 35 masks per target while *IMDb* forces, in most cases, the computation

of several hundreds masks. We remark that these datasets have very different original size which confirms that the complexity of the attack is only marginally linked to the total size of the dataset.

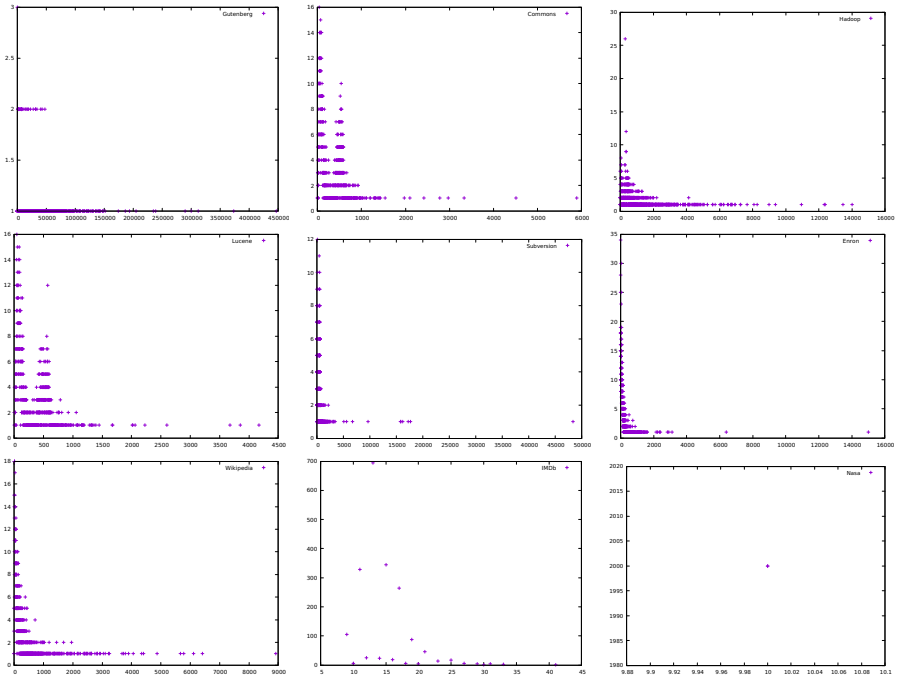

Fig. 2. Number of documents depending on the length with the L4 view.

Since the governing parameter for the cost of the CoMask attack is also the number of candidate of equal length, we perform a very similar analysis but this time considering the length in view of the L3 leakage profile, i.e., we count the number of different keywords in each document. Since the length under this leakage profile is necessarily smaller than the length in the view of the L4 leakage profile we expect to see similar results with more "collisions" of length. This is indeed what we observe in figure 3. Both extremes stay the same: there is at most 4 candidates for each target plaintext on the *Gutenberg* dataset and again almost all documents of the *Nasa* dataset have the same length. In between, *Commons*, *Hadoop*, *Lucene*, *Subversion*, *Enron*, and *Wikipedia* have at most 40 candidate per target while *IMDb* boasts again several hundreds candidates. We note here that while in the case of the mask attack the cost grows linearly with the number same-length candidates, the cost of the CoMask attack grows like its square (since we compute a pair of matching co-resulting mask). This gives

the intuition that the attacks could be made much slower by simply padding all the encrypted documents.

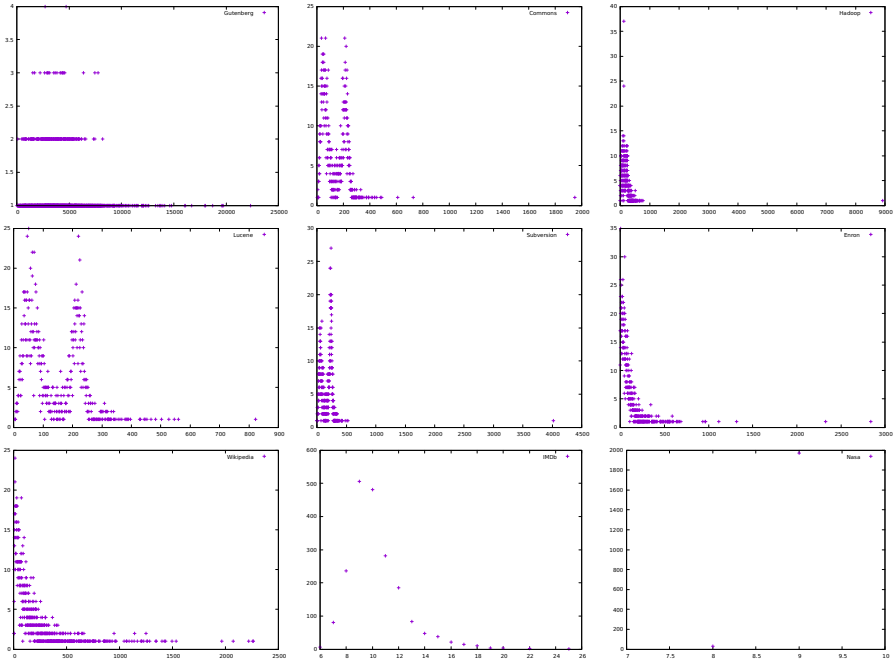

Fig. 3. Number of documents depending on the length with the L3 view.

Finally, we measure the occurrence of all keywords present in the sample, i.e., the number of documents containing the considered keyword, starting from the less frequent and ending with the most frequent. Results are presented in Figure 4 and help to predict the efficiency of the PowerSet attack. In fact, the PowerSet attack is based on the number of keywords shared between documents. Hence, if a dataset has a lot of keywords shared between its documents, then the probability to distinguish documents on their shared documents is higher. We can predict that the PowerSet attack will more efficient on *Gutenberg*, *Commons*, *Hadoop*, *Lucene*, *Subversion*, *Enron*, *Wikipedia* datasets, than on *IMDb* and *Nasa* datasets.

5.3 Efficiency Measures

We ran our attacks for different sizes of $\mathcal{S}$ using steps of 1% until 10% then steps of 10% from 10% to 100%. Here 1% is 1% of the pairs (d, w) of the dataset; this allows us to perform a fairer comparison between datasets than the usual per-

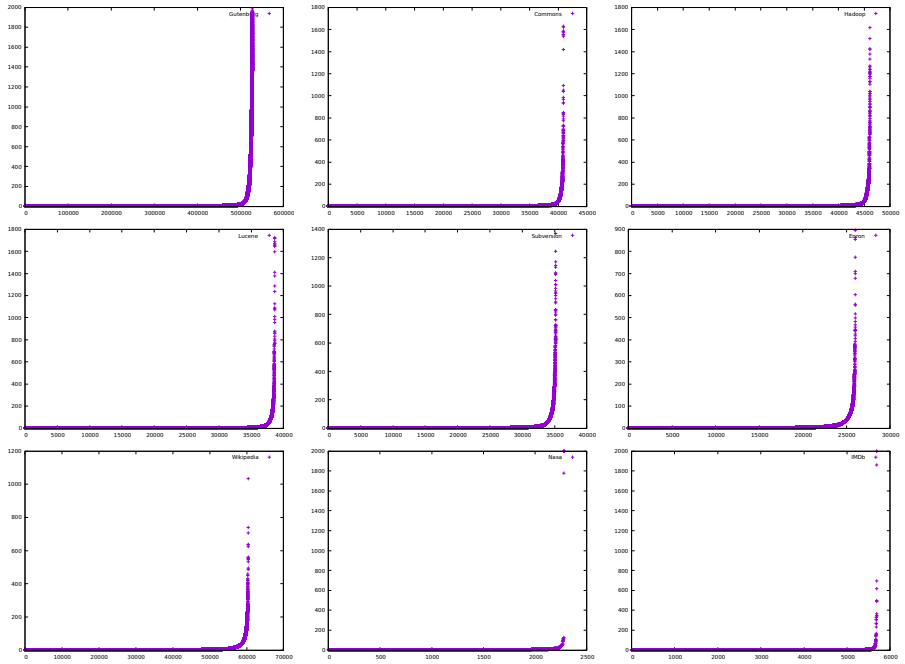

Fig. 4. Occurrence of keywords, from less frequent to more frequent.

document measure, as knowing a long document do not have the same impact as knowing a short one.

The measured *success rate* is the ratio of keywords-tokens associations over the set of keywords of $\mathcal{S}$. Then, these correspondences are spread back into EDB in order to evaluate their impact on other documents of the dataset. In particular, we measured the rate of documents of the dataset whose keywords are recovered at 80%, 90% and 100%.

5.4 Experimental Results on Datasets Samples

We expose here the results of our attacks on the chosen datasets. All attacks are performed on an Intel® Xeon® using 64 Gb RAM with REDIS[9], an in-memory data structure store.

Experimental Results of the Mask Attack. Results for the Mask attack are presented in Figure 5. We notice that we have three categories of results.

The first category is when the mask attack can associate with a high ratio keywords and tokens. As shown in Figure 7, this is the case for *Gutenberg*,

[9] https://redis.io/

Commons, *Hadoop*, *Lucene*, *Subversion*, *Enron*, *Wikipedia* datasets. In fact, for each percentage of knowledge, the adversary can associate with a ratio equals to 1, keywords that it knows to their corresponding tokens. Hence, we can observe that the Mask attack is ravaging for mailing list, emails, articles, and books datasets.

Second, very few associations between keywords and tokens can be done. As exposed in Figure 5, the Mask attack result on dataset *IMDb* is in this category. As we can see in Figure 2, a lot of documents have the same length. This is due to the structure of the dataset. In fact, movies descriptions are very similar since each keyword has a specific set of possible values. However, the number of keywords composing the title of the movie can be discriminant that is why we can associate some documents with the Mask attack. However, the adversary can at most associate 10% of the keywords that it knows to their corresponding tokens.

Finally, no association between keywords and tokens can be performed whatever the part of knowledge over the dataset. As we can see in Figure 5, results of the Mask attack on the *Nasa* dataset is in this category. Each server log is composed of the same number of keywords, hence mask of server logs cannot be distinguishable by the length (Figure 2). Moreover, each keyword in a server log has a specific role, for instance: client IP address, request date/time, page requested, HTTP code, bytes served, then the probability to have repetitions in the masks (for example, hostname equals to the timestamp or to the number of bytes served) is very low. Hence, masks of server logs are always the same, i.e., $(1, \ldots, \ell)$ where ℓ is the fixed number of keywords in server logs. This is why our mask attack is inefficient against server logs dataset, i.e., the adversary can not associate keywords that it knows to their corresponding tokens.

Experimental Results of the CoMask Attack. Results for the CoMask attack are presented in Figure 6. We notice that we have two categories of results.

The first category is where the ratio of association between keywords and tokens is high for each percentage of known dataset. As shown in Figure 6, datasets *Gutenberg*, *Commons*, *Hadoop*, *Lucene*, *Subversion*, *Enron* and *Wikipedia* are in this category since the adversary can associate almost 100% of the keywords in the known dataset to their corresponding tokens. Their common particularity is that they have a lot of frequent keywords (Figure 4). In fact, the CoMask attack uses the shared keywords between documents to distinguish computed comask. Hence, we argue that datasets having a such distribution of frequent keywords are less resistant against the CoMask attack since the documents pairs can be more distinguishable.

The second category concerns *Nasa* and *IMDb* datasets. As we can see in Figure 6, the ratio of keyword-token associations is very low (3% for the *Nasa* dataset, and 10% for the *IMDb* dataset). With Figure 4, we see that these two datasets have very few frequent keywords. Hence, computed comask from the

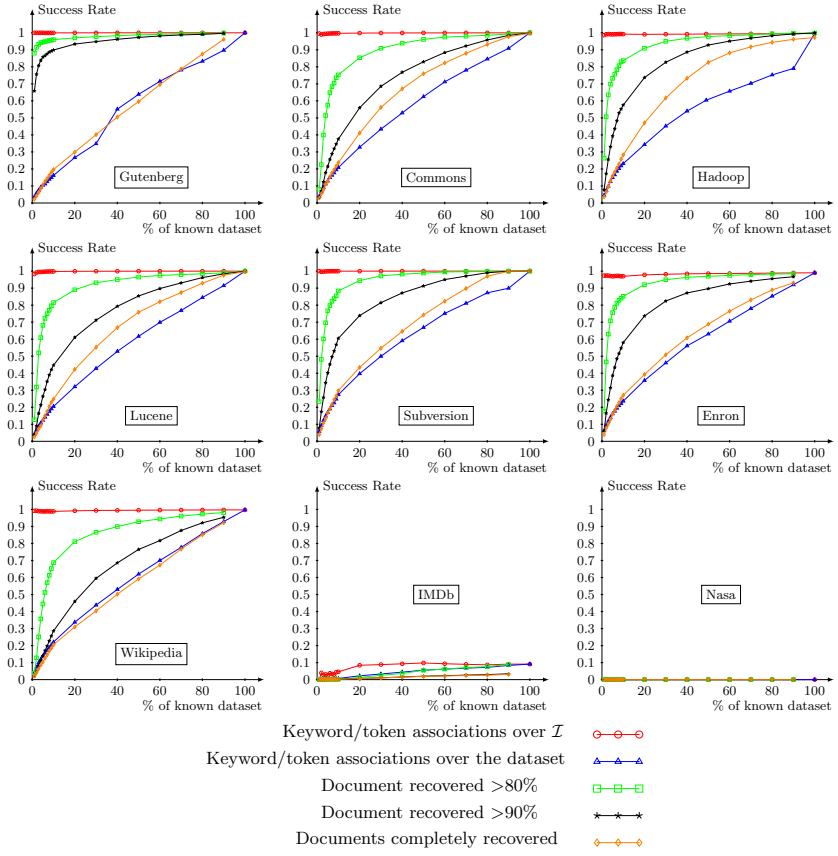

Fig. 5. Efficiency of our *Mask* attack depending on the knowledge rate of the server.

percentage of known dataset do not discriminate. We assume that a dataset having very few shared frequent keywords is resistant to the CoMask attack.

Experimental Results of the PowerSet Attack. Results for the PowerSet attack are presented in Figure 7. We notice that we have four categories of results.

First, the ratio of association between keywords and tokens is always high whatever the part of knowledge of the dataset. It is the case for the PowerSet attack applied to the *Gutenberg* dataset as shown in Figure 7 where an adversary can always associate keywords to their corresponding tokens whatever the percentage of known dataset. As shown in Figure 4, the occurrence of frequent keywords is much bigger in the *Gutenberg* dataset, i.e., a lot of documents in the dataset share different keywords. In the context of the PowerSet attack, this

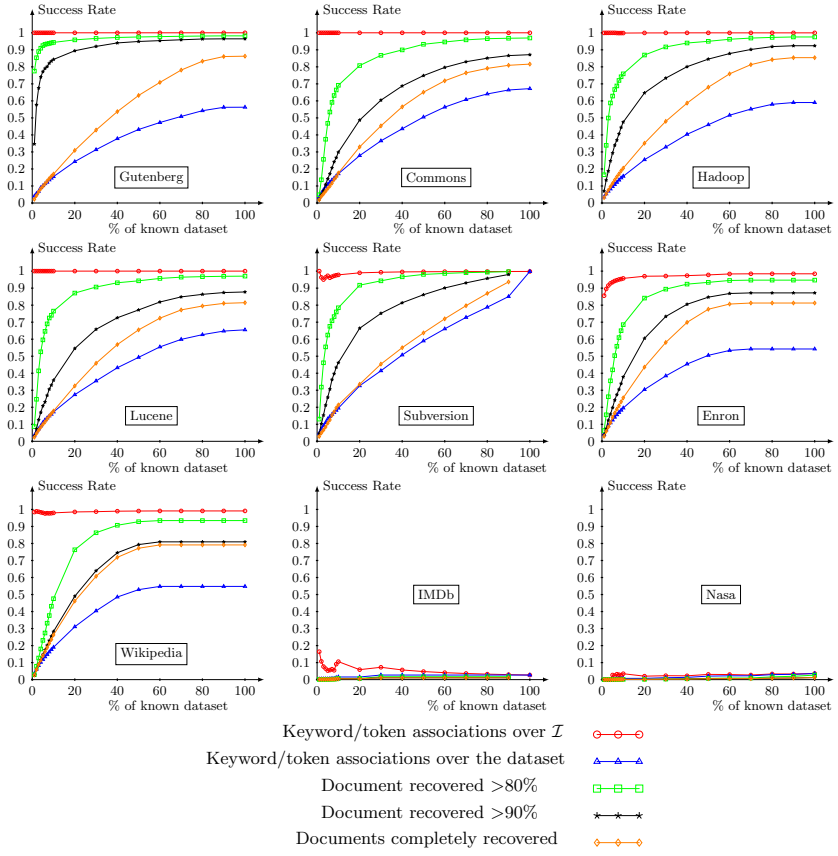

Fig. 6. Efficiency of our *CoMask* attack depending on the knowledge rate of the server.

fact helps to match known documents to their representation in the encrypted documents, and to recover keywords.

The second category of results is illustrated in Figure 7 by *Commons*, *Hadoop*, *Lucene*, and *Subversion* datasets. In this case, the association between keywords and tokens depending of the part of knowledge of the dataset is relatively of a constant order. As shown in Figure 4, the most frequent keywords of these four datasets are present in a quarter of the documents while the most frequent keywords are present up to five times per document. We assume that a dataset with a such distribution for keywords occurrences is sensible to the PowerSet attack.

Third, the association between keywords and tokens is progressive depending the part of knowledge of the dataset. We see in Figure 7 that results of the PowerSet attack on *Enron* and *Wikipedia* datasets are in this category. The similarity between these two datasets lies in the distribution of the mots frequent

keywords. As we can see in Figure 4, frequent keywords are present in a quarter of documents while the most frequent keyword is in all the documents or twice. Due to the nature of the PowerSet attack, we assume that a dataset where most frequent keywords have a such distribution is sensible to the PowerSet attack since the knowledge of keyword-token association is proportional to the part of knowledge of the dataset.

Finally, no association between keywords and tokens can be performed, and this whatever the part of knowledge over the dataset. As we can see in Figure 7, results of the PowerSet attack on *Nasa* and *IMDb* datasets are of this type since an adversary can not associate keywords to their corresponding tokens. These two datasets have a small keywords space: 2,280 for *Nasa* dataset and 5,671 for *IMDb* dataset. Moreover, we see in Figure 4 that keywords occurrences in these datasets are more homogeneous than in the other datasets. Since, the PowerSet attack deals with the number of shared keywords between documents, we deduce empirically that a dataset with a small keywords space and with a homogeneous distribution of keywords occurrences is resistant against the PowerSet attack.

Remark. As noted in [9], this reconstruction allows to reveal sensitive information even if the order of keywords is not preserved. Human inspection of the output of our attacks gives a clear idea of the sense of each document.

5.5 Scaling up: Experimental Results on Full Datasets

In order to get a more complete picture of the practicality of the attacks, we present in this section results of the Mask, CoMask, and PowerSet attacks on three full datasets: *Commons*, *Lucene*, and *Hadoop*. The results of the attack themselves are very close to the ones presented in the previous section: the details on these datasets are given in Fig. 8. The benefit of running the attacks on the full datasets was the impact of low probability events (only observable on large documents sets) on the running time of the attacks. In particular two phenomenon slowed down the attacks majorly and had to be dealt with in a specific way. The first of these phenomenons was the presence of duplicates in the original datasets. If not handled specifically, duplicates documents actually prevent "early-abort" in both the CoMask and the Powerset attacks. Unfortunately at high orders (several hundreds of documents) one cannot afford to not quickly eliminate candidates. This has the practical consequence that duplicates need to be removed before the elimination procedure which incurs a cost that grows like the square of the number of same-length candidate. The second phenomenon is similar but worse in its consequences. There are documents that are indistinguishable in the sense that they are almost identical but their difference appear in the same documents. Actually, the case most often encountered is documents that are equal up to 2 words (one in each document) but that those 2 words do not appear anywhere else in the dataset. That means that the two documents are not equal (and therefore would not be filtered out by the procedure outline above) but they are effectively equivalent through the leakage

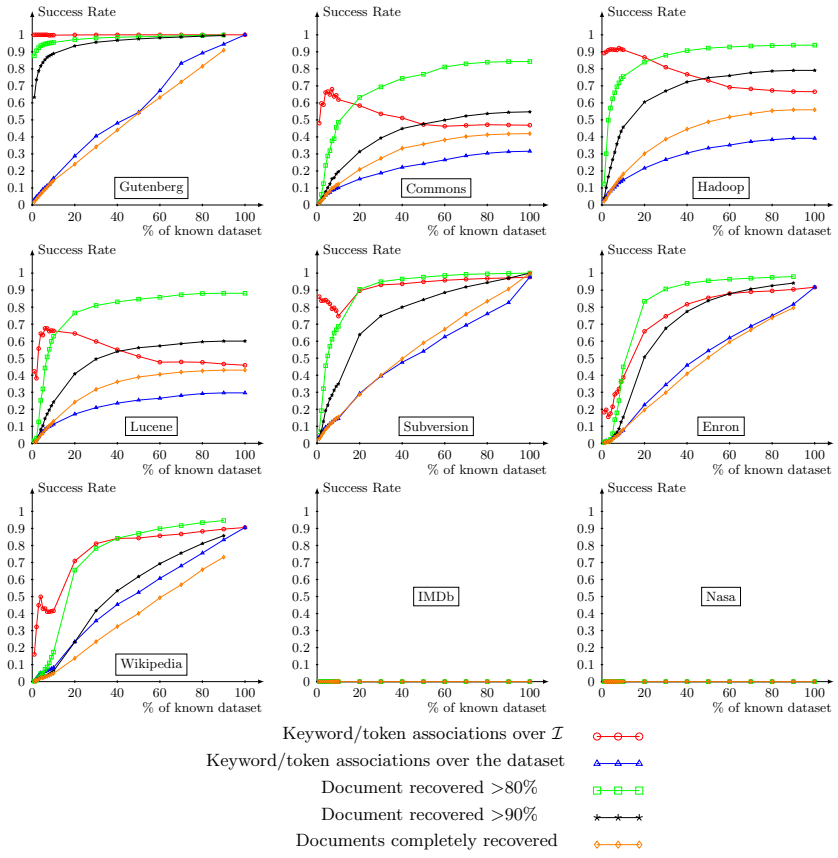

Fig. 7. Efficiency of our *PowerSet* attack depending on the knowledge rate of the server.

profile. Dealing with those documents during the attacks incurs a very high cost even in comparison to the task of eliminating duplicates.

Dataset	Content	#DB	#W	N
Commons	mailing list	28,997	230,893	3,910,562
Lucene	mailing list	58,884	394,481	7,952,794
Hadoop	mailing list	21,312	206,315	3,655,222

Fig. 8. Details on selected full datasets.

Results of the Mask attack are presented in Fig. 9, results of the CoMask attack in Fig. 10, and results of the PowerSet attack are exposed in Fig. 11.

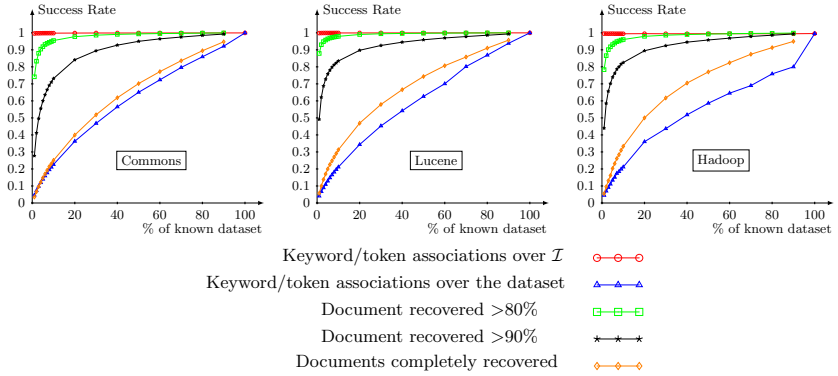

Fig. 9. Efficiency of our *Mask* attack depending on the knowledge rate of the server.

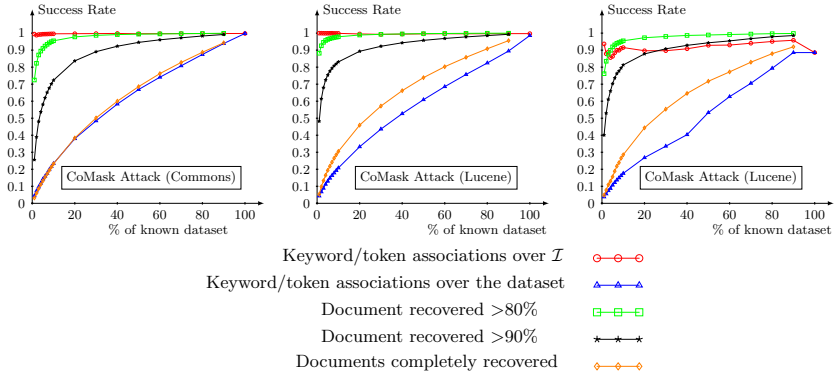

Fig. 10. Efficiency of our *CoMask* attack depending on the knowledge rate of the server.

5.6 Further Protecting Your Datasets

The previous sections have given some intuition as to what would constitute the most promising avenue for efficient countermeasures against leakage-abuse attacks. We leave as future work the analysis of the cost/benefit ratio of those countermeasures but any practical use of SSE scheme should consider implementing the following counter-measures.

Length padding. The owner of the dataset should make sure that all of the documents in the datasets have a least $t-1$ other documents of the same length with t being large enough that t^2 calculations of co-resulting masks (respectively, powersets of order 2) is considered prohibitive.

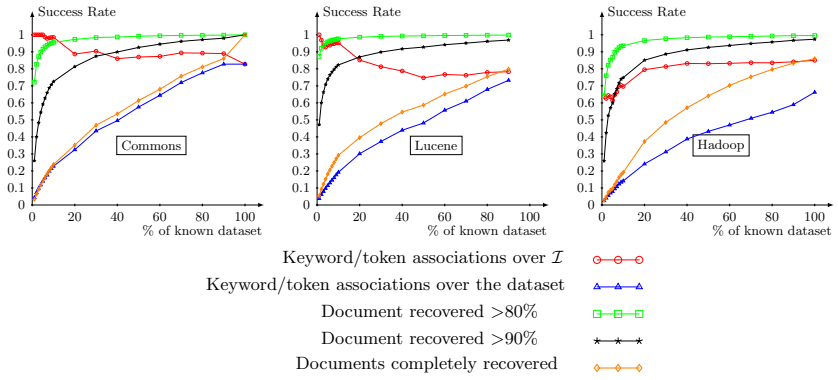

Fig. 11. Efficiency of our *PowerSet* attack depending on the knowledge rate of the server.

Ghost twin insertion. The owner of the dataset should insert "ghost twin documents" in the dataset. The "ghost twin" of a document d is a document d' that is indistinguishable from d through L2 leakage profile. The easiest method for creating ghost twins is to start from the inverted index with a keyword w that only appears in one document d. The ghost twin of d is the document d where all the occurrences of w have been replaced by $\tilde{w}$ and where $\tilde{w}$ does not appear anywhere else in the dataset.

6 Conclusion

SSE schemes are known to be insecure considering an adversary who is able to perform file-injections. In fact, Zhang et al. [28] shown that only few injected files can reveal to an adversary associations between keywords and search tokens. Moreover, SSE schemes are also vulnerable to passive observations of search tokens since they reveal the underlying searched keyword when the data set is completely known as shown in [9, 20, 25].

In this paper, we focus on the impact of passive attacks on SSE schemes of profiles L4, L3 and L2, i.e., families of SSE schemes that are currently used as commercially solutions [2, 1, 3]. We consider an adversary who has access to the encrypted database (stored by the server) and knows a sample of plaintexts having generated this encrypted database. As we shown first in [16], our attacks are devastating on most real-world datasets that we used,namely mailing-list, emails, and books datasets. Indeed, regardless of the leakage profile and knowing a mere 1% of the original documents in data set, we are able to recover 80% of the content of 90% of the documents. However, the efficiency of our attacks depends on the nature of the considered data set. In fact, as we shown in this paper, an adversary performing our attacks on encrypted databases generated from datasets composed of very formatted data like server logs or of movies de-

scriptions extracts not much information on original documents. In the best case, she can recover 80% of the content of only 5% of the documents. In fact, such documents are very structured and contain very few shared words, preventing the retrieval of information to associate keywords and tokens, thus avoiding the retrieval of the content of the documents. Finally we gave insight into interesting techniques for further protecting encrypted databases based on the knowledge acquired during our experiments

To conclude, the results in this paper does cast doubt on the legitimacy of SSE schemes and help us to better understand the practical security of SSE schemes. Relying on the experimental results of our attacks on the server logs and movies descriptions data sets, it will be interesting to further investigate the cost/benefit ratio of countermeasures to these attacks.

Acknowledgments

This research was conducted with the support of the FEDER program of 2014-2020, the region council of Auvergne-Rhône-Alpes, the support of the "Digital Trust" Chair from the University of Auvergne Foundation, the Indo-French Centre for the Promotion of Advanced Research (IFCPAR) and the Center Franco-Indien Pour La Promotion De La Recherche Avancée (CEFIPRA) through the project DST/CNRS 2015-03 under DST-INRIA-CNRS Targeted Programme.

References

1. Bitglass. Security, Compliance, and Encryption. http://www.bitglass.com/salesforce-security. Accessed: 2017-01-18.
2. CipherCloud. Cloud Data Encryption. https://www.ciphercloud.com/technologies/encryption/. Accessed: 2017-01-18.
3. Skyhigh Networds. Skyhigh for Salesforce. https://www.skyhighnetworks.com/product/salesforce-security/. Accessed: 2017-01-18.
4. M. A. Abdelraheem, T. Andersson, C. Gehrmann, and C. Glackin. Practical attacks on relational databases protected via searchable encryption. *IACR Cryptology ePrint Archive*, 2018:715, 2018.
5. Apache Commons email dataset. http://mail-archives.apache.org/mod_mbox/commons-user/. Accessed: 2016-04.
6. Apache Hadoop email dataset. http://mail-archives.apache.org/mod_mbox/hadoop-user/. Accessed: 2018-03.
7. Apache Lucene email dataset. http://mail-archives.apache.org/mod_mbox/lucene-java-user/. Accessed: 2016-04.
8. Apache Subversion email dataset. http://mail-archives.apache.org/mod_mbox/subversion-users/. Accessed: 2018-03.
9. D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-Abuse Attacks Against Searchable Encryption. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 668–679, New York, NY, USA, 2015. ACM.

10. D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.
11. D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 353–373, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
12. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 79–88, New York, NY, USA, 2006. ACM.
13. Enron email dataset. http://www.cs.cmu.edu/~./enron/. Accessed: 2016-04.
14. S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich Queries on Encrypted Data: Beyond Exact Matches. In *Computer Security – ESORICS 2015: 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, pages 123–145. Springer International Publishing, 2015.
15. C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.
16. M. Giraud, A. Anzala-Yamajako, O. Bernard, and P. Lafourcade. Practical Passive Leakage-abuse Attacks Against Symmetric Searchable Encryption. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT, Madrid, Spain, July 24-26, 2017.*, pages 200–211, 2017.
17. O. Goldreich. *Secure Multi-party Computation*, 1998. Working Draft.
18. Project Gutenberg. http://www.gutenberg.org/wiki/Main_Page. Accessed: 2016-04.
19. IMDb dataset. https://www.imdb.com/interfaces/. Accessed: 2018-03.
20. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.
21. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic Searchable Symmetric Encryption. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 965–976, New York, NY, USA, 2012. ACM.
22. B. Lau, S. P. Chung, C. Song, Y. Jang, W. Lee, and A. Boldyreva. Mimesis aegis: A mimicry privacy shield-a system's approach to data privacy on public cloud. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 33–48, 2014.
23. Nasa-HTTP server logs. http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html. Accessed: 2018-03.
24. M. F. Porter. An algorithm for suffix striping. *Program*, 14(3):130–137, 1980.
25. D. Pouliot and C. V. Wright. The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1341–1352, 2016.

26. D. X. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 44–, Washington, DC, USA, 2000. IEEE Computer Society.
27. G. Wang, C. Liu, Y. Dong, K. K. R. Choo, P. Han, H. Pan, and B. Fang. Leakage Models and Inference Attacks on Searchable Encryption for Cyber-Physical Social Systems. volume PP, 2018.
28. Y. Zhang, J. Katz, and C. Papamanthou. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. Cryptology ePrint Archive, Report 2016/172, 2016. http://eprint.iacr.org/2016/172.