Practical Passive Leakage-Abuse Attacks Against Symmetric Searchable Encryption

Matthieu Giraud[1], Alexandre Anzala-Yamajako[2], Olivier Bernard[2], and Pascal Lafourcade[1]
[1] *Université Clermont Auvergne, BP 10448, F-63000, Clermont-Ferrand, France.*
{*firstname.lastname*}*@uca.fr*

[2] *Thales Communications & Security, 4 avenue des Louvresses, 92622, Gennevilliers, France.*
{*alexandre.anzalayamajako,olivier.bernard2*}*@thalesgroup.com*

Keywords: Symmetric Searchable Encryption, Leakage, Passive Attacks.

Abstract: *Symmetric Searchable Encryption* (SSE) schemes solve efficiently the problem of securely outsourcing client data with search functionality. These schemes are provably secure with respect to an explicit leakage profile; however, determining how much information can be inferred in practice from this leakage remains difficult. First, we recall the leakage hierarchy introduced in 2015 by Cash et al. Second, we present complete practical attacks on SSE schemes of L4, L3 and L2 leakage profiles which are deployed in commercial cloud solutions. Our attacks are passive and only assume the knowledge of a small sample of plaintexts. Moreover, we show their devastating effect on real-world data sets since, regardless of the leakage profile, an adversary knowing a mere 1% of the document set is able to retrieve 90% of documents whose content is revealed over 70%. Then, we further extend the analysis of existing attacks to highlight the gap of security that exists between L2- and L1-SSE and give some simple countermeasures to prevent our attacks.

1 INTRODUCTION

With the growing importance of digital data in everyday life, it is necessary to have backups and to have access from anywhere. For these reasons, outsourcing this digital data to a cloud provider is an enticing solution. However, some of this data, such as legal documents, banking and medical, the industrial patents or simply our emails can be sensitive and/or confidential, forcing the user to trust its cloud provider. Client-side symmetric encryption is the classical answer to the problem of data confidentiality. However, encryption prevents any server-side processing of the client data as is the norm on plaintext data. In particular, a server is not able to answer search queries, that is given a keyword, retrieve the documents containing that keyword. *Symmetric Searchable Encryption* (SSE) schemes introduced in (Song et al., 2000) aim at retaining this search capability on encrypted data. SSE scheme is a protocol between a client and a server. The client owns a sensitive data set but has limited computational power and storage capacity. The server has a large storage space and high processing power, but is not trusted by the client except for executing correctly the search protocol. The set of plaintext documents are stored in a *DataBase* (DB). An SSE scheme creates metadata that is protected in an *Encrypted DataBase* (EDB) and then stored by the server. From a keyword and his symmetric secret key the client creates a search token that is sent to the server who finds the encrypted documents matching the query with the help of EDB. Such documents are then sent back to the client for decryption. While the single keyword query is the basic functionality of an SSE scheme there exist SSE schemes which allow the client to add new encrypted documents to the encrypted database while retaining the search capability (Cash et al., 2014; Kamara et al., 2012) and others which focus on expanding the expressiveness of the search queries such as Boolean (Cash et al., 2013) and sub-string search queries (Faber et al., 2015).

The amount of information leaked by a given SSE scheme to the server is formalized by a *leakage function* (Curtmola et al., 2006; Kamara et al., 2012). The security of the scheme then relies on proving that this function does not leak more information than expected. However, it can be used by an honest-but-curious server (Goldreich, 1998), which dutifully executes the scheme but tries to deduce information on the stored documents. By its nature, a SSE scheme

reveals to an observer the *search* and the *access* pattern. In fact, a client searching twice the same keyword sends the same query. And so, the server replies to these queries in the same way. These search and access patterns are used in *inference attacks* (Islam et al., 2012; Cash et al., 2015; Pouliot and Wright, 2016) whereas our passive attacks do not use these information. In this paper, we focus on the information revealed by the encrypted database regardless exchanges between the client and the server. This model assumes that the adversary can be the server himself or a malicious person who is able to access to the encrypted database stored on the server. Based on deployed SSE schemes, (Cash et al., 2015) define four *leakage profiles* L4, L3, L2 and L1, L4 being the most leaky and L1 the least. Commercially available SSE solutions are L4 schemes such CipherCloud* and Skyhigh Networks†, L3 schemes such Bitglass‡ or L2 schemes as ShadowCrypt (He et al., 2014) and Mimesis (Lau et al., 2014) while proposed schemes in academic research are L1 schemes. L4-, L3- and L2-SSE schemes can be used as a proxy for existing cloud solutions or as extensions in client-side and so do not require any modification on server-side. Assessing the practical impact of each of these profiles on the server knowledge of the protected data is critical for real life applications. We study the impact of a passive attacker.

Our contributions. Only assuming the knowledge of a small sample of plaintexts in addition to the protected database given to the server, we design passive attacks on L4, L3 and L2 leakage profiles. In particular, our attacks do not rely on observing search queries. Our attacks exploit the leaked information the scheme on the encrypted database to find, starting from a sample of plain documents, their identifiers in the encrypted database. Then, knowing these correspondences, the adversary tries to determine values of plain keywords in the encrypted database to recover other documents. 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. The attack on L2 schemes Their efficiency and practicality are demonstrated on several real-world data sets such as the mailing-list of *Lucene* Apache project§. In fact, the knowledge of a small sample of plain documents by an adversary has a huge impact. With our passive attacks on L4- and L3-SSE schemes, an adversary knowing only 1% of plain documents is able to reconstruct 90% of the protected data at 80%. For our passive attack on L2-SSE schemes, the knowledge of 1% of plain documents implies the recovering of 70% the protected data at 80%. In this paper, we also deal with the gap of security that exists between L2- and L1-SSE schemes in depth and show that L1-SSE are much more robust against passive attacks while client do not perform many queries. Finally, we propose trails of countermeasures for our attacks. Countermeasures for our attacks on L4- and L3-SSE schemes are efficient since no information can be deduced although the adversary knows a sample of plain documents. Moreover, they generate not many false positives. On the contrary, the countermeasure for our attack on L2-SSE schemes is generic but requires a not negligible precomputing phase and generates more false positives.

Related Work. For an *active* adversary able to plant chosen documents in the database, (Cash et al., 2015) present a partial document recovery attack on L3- and L2-SSE schemes. With the extra ability to issue selected queries, (Zhang et al., 2016) mount a query recovery attack that works on any dynamic SSE scheme. These active attacks are very efficient as few injected files reveal associations between keywords and search tokens but are different from ours since we consider only a passive adversary who is not able to plant document in the database.

Inference attacks based on the observation of client queries and server responses have been also proposed. The first one is the *IKK Attack*, proposed in (Islam et al., 2012). Its goal is to associate search tokens to actual keywords, exploiting the data access pattern revealed by client queries and assuming the adversary has access to a co-occurrence matrix that gives the probability for two keywords to appear in a randomly chosen document. As noted in (Cash et al., 2015), this matrix needs to be so precise for the attack to succeed, that it seems legitimate to suppose the adversary has access to the number of documents in which every keyword appears. With this strong extra knowledge, they mount a more effective attack named the *Count Attack* (Cash et al., 2015). Both attacks target leakage profiles beyond L1, but the strength of their assumptions questions their practicality. In comparison our attacks do not rely on observing client queries but only consider the encrypted database as viewed by the adversary. We compare IKK and Count attacks to our passive PowerSet attack in Section 6.

Additionally, (Cash et al., 2015) propose a passive partial document recovery attack for L3-SSE schemes when the adversary knows plaintext-ciphertext pairs. Our attacks suppose that we have not plaintext-

*ciphercloud.com/technologies/encryption/

†skyhighnetworks.com/product/salesforce-security/

‡bitglass.com/salesforce-security

§mail-archives.apache.org/mod_mbox/lucene-java-user/

ciphertext pairs initially. An other approach, called *Shadow Nemesis Attack*, is proposed in (Pouliot and Wright, 2016). Using a training data set, this inference attack builds a co-occurrence matrix and reduce the problem of matching search tokens to keywords to the combinatorial optimization problem of weighted graph matching. This attack can be performed on L2-SSE schemes as our attacks. It uses the encrypted database and a training data set or partial knowledge on the original data set whereas our attacks use only partial knowledge on the original data set. We show in Section 6 that our PowerSet attack recovers more keywords with the same knowledge.

Outline. In Section 2, we provide background on SSE schemes and their security. We recall in Section 3 the leakage hierarchy of (Cash et al., 2015). We describe our new passive attacks in Section 4 and demonstrate their effectiveness in Section 5. We show in Section 6 the gap for an adversary to recover client queries between L2- and L1-SSE schemes and give countermeasures for our attacks in Section 7.

2 SYMMETRIC SEARCHABLE ENCRYPTION

We introduce notations, then we formalize SSE schemes and discuss the associated security notion.

Sequences, lists and sets. A sequence of elements is defined as an ordered set where repetitions are allowed. A list is an ordered set where all elements are distinct. A set is defined as a bunch of distinct elements with no order. Sequences are guarded by $(\dots)$, lists are denoted by square brackets $[\dots]$ and sets by braces $\{\dots\}$. The number of elements of a set E (resp. list or sequence) is written $\#E$.

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

The same objects are described server-side by introducing the star superscript. Hence, $\mathsf{W}^* = \{w_1^*, \dots, w_m^*\}$ denotes the set of search tokens associated to the keywords of W. Similarly, $\mathsf{DB}^* = \{d_1^*, \dots, d_n^*\}$ is the set of ciphertexts of DB where d_i^* is the encryption of d_i, and W_i^* is the set of tokens associated to d_i^*. As to emphasize the fact that the association between d_i and d_i^* is not known to the server a priori, an identifier id_i is used to uniquely represent d_i^*. A data structure EDB is also provided, which contains protected metadata that allows the server to answer search queries.

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

2.1 Static SSE Schemes

Basic SSE schemes are defined by a symmetric encryption scheme together with an algorithm for setup and another for search.

As a first step, the client creates two data structures DB^* and EDB as introduced above. Both data structures are then uploaded to the server. As a second step, when the client wants to search for a specific keyword w, he computes the associated search token w^* with his secret key and sends w^* to the server. From w^* and EDB the server is able to return the identifiers of all encrypted documents matching the client's search. With the list of identifiers the client retrieves the encrypted documents, from which he can 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.

Definition *Static* SSE *scheme.* Given a symmetric encryption scheme $(\mathcal{E}.(\cdot), \mathcal{D}.(\cdot))$ where $\mathcal{E}.(\cdot)$ denotes the encryption algorithm and $\mathcal{D}.(\cdot)$ denotes the decryption algorithm, 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 algorithm 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 run by the client to set the scheme up. It takes as input secret keys K and k, the database DB and the encryption algorithm $\mathcal{E}.(\cdot)$, and outputs both the protected metadata EDB and the encrypted documents $\mathsf{DB}^* = (\mathcal{E}_k(d_1), \dots, \mathcal{E}_k(d_n))$.

$w^* \leftarrow \mathsf{SearchClient}(K,w)$ is a deterministic algorithm 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 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.

This defines *static* SSE schemes. Static SSE schemes allow the client to initialize a protected database that supports keyword searches but cannot be updated by opposition to *dynamic* SSE schemes. We do not introduce dynamic schemes since their encrypted databases can be attacked at least as well as static schemes.

2.2 Security of SSE Schemes

Introduced by Curtmola et al. in (Curtmola et al., 2006) and by Kamara et al. in (Kamara et al., 2012), the *leakage function* $\mathcal{L}$ of a SSE scheme is a set of information revealed by the SSE scheme to the server. This leakage function formalizes information that EDB and the client queries reveal to the server.

The SSE scheme is said to be $\mathcal{L}$-secure if and only if any polynomial-time adversary making a sequence Q of queries (i.e. keywords of W) can successfully tell with only negligible probability whether the protocol is honestly executed or simulated from the leakage function $\mathcal{L}$. The $\mathcal{L}$-security proves that no information is leaked by the SSE scheme to the server outside of what is exposed by the leakage function. We focus on the *practical* impact on the knowledge of the protected data.

3 A LEAKAGE HIERARCHY

We recall classes of SSE schemes according to how much information the protected database leaks, as first introduced in (Cash et al., 2015).

$\mathsf{L4}$ **Leakage Profile.** Without any semantic consideration, a document is characterized by its number of words, their order and their occurrence counts. Moreover, it is possible to know which words are shared with any other document. $\mathsf{L4}$-SSE schemes used by commercial encryption products as CipherCloud reveal these information, so nothing is lost about the plaintext non-semantic structure. A SSE scheme of leakage function $\mathcal{L}$ is of class $\mathsf{L4}$ if and only if $\mathcal{L}(\mathsf{EDB}) = \left\{(w^*_{i_1}, \ldots, w^*_{i_{\ell_i}})\right\}_{1 \leqslant i \leqslant n}$.

Example. We use the following setup as a running example to illustrate the different amounts of leakage revealed to the server. Let $\mathsf{W} = \{\text{as}, \text{call}, \text{i}, \text{if}, \text{me}, \text{possible}, \text{soon}, \text{you}\}$ and d_1 and d_2, two documents defined over W where $d_1 = (\text{call}, \text{me}, \text{as}, \text{soon}, \text{as}, \text{possible})$ and $d_2 = (\text{i}, \text{call}, \text{you}, \text{if}, \text{possible})$. Assume that the search tokens W^* associated to keywords of W are the following:

W	W^*
as	14
call	76
i	33
if	11

W	W^*
me	25
possible	35
soon	78
you	10

Under $\mathsf{L4}$ leakage, EDB reveals to the server $(76, 25, 14, 78, 14, 35) \rightarrow \mathsf{id}_1$ and $(33, 76, 10, 11, 35) \rightarrow \mathsf{id}_2$. The server knows that the document identified by id_1 is of length 6 and has five distinct keywords; it also knows that one keyword, associated to the token 14, is repeated twice. The document identified by id_2 contains five distinct keywords and shares two keywords with the first document represented by tokens 35 and 76.

$\mathsf{L3}$ **Leakage Profile.** For keyword search purposes, it is not necessary to know the occurrence count of each keyword. Then a SSE scheme of leakage function $\mathcal{L}$ is of class $\mathsf{L3}$ if and only if $\mathcal{L}(\mathsf{EDB}) = \left\{\mathsf{L3}_{\mathsf{EDB}}(\mathsf{id}_i)\right\}_{1 \leqslant i \leqslant n}$, where $\mathsf{L3}_{\mathsf{EDB}}(\mathsf{id}_i) = \left[(w^*_{i_1}, \ldots, w^*_{i_{\ell_i}})\right]$.

Example. Resuming the running example, the information revealed by an $\mathsf{L3}$-SSE scheme about d_1 and d_2 is: $(76, 25, 14, 78, 35) \rightarrow \mathsf{id}_1$ and $(33, 76, 10, 11, 35) \rightarrow \mathsf{id}_2$. The server does not know anymore that the token 14 is associated twice to id_1.

$\mathsf{L2}$ **Leakage Profile.** $\mathsf{L2}$-SSE schemes, as (He et al., 2014), only reveal the set of tokens of a document. The server can still determine which documents contain a given token. A SSE scheme of leakage function $\mathcal{L}$ is $\mathsf{L2}$ if and only if $\mathcal{L}(\mathsf{EDB}) = \left\{W^*_i\right\}_{1 \leqslant i \leqslant n}$.

Example. Resuming the running example, an $\mathsf{L2}$-SSE scheme reveals about d_1 and d_2: $(14, 25, 35, 76, 78) \rightarrow \mathsf{id}_1$ and $(10, 11, 33, 35, 76) \rightarrow \mathsf{id}_2$. We stress that the token order is not preserved in EDB: we arbitrarily sorted the token in ascending order, thus the server does not know their initial order.

$\mathsf{L1}$ **Leakage Profile.** With no initial search, $\mathsf{L1}$-SSE schemes, as (Cash et al., 2014; Curtmola et al., 2006), leak the least possible amount of information, i.e. the number N of document/keyword pairs of the data set. Thus $\mathcal{L}(\mathsf{EDB}) = \{N\}$.

Example. Resuming the running example, the information revealed by an $\mathsf{L1}$-SSE scheme looks like:

w^*	α	β	γ	δ	ε	ζ	η	θ	ι	κ
Id.	a	b	c	d	e	f	g	h	i	j

Greek (resp. Latin) letters represent tokens (resp. identifiers). The server has absolutely no clue about this correspondence, so it only knows $N = 10$. If the client searches for "soon" and "you", this reveals:

w^*	α	β	γ	δ	ε	35	35	θ	ι	78
Id.	a	b	c	d	e	id_1	id_2	h	i	id_1

Hence, the server learns that documents identified by id_1 and id_2 share the same keyword of token 35; keywords of tokens 14 and 35 are both in the document identified by id_1.

Effect of Queries on the L1 Leakage Profile. We study what can be inferred from the protected database, but it is informative to reflect upon the effect of queries on the amount of information revealed to the server. At the end of the search protocol the client obtains identifiers of the documents matching its query. Server-side this can be leveraged to associate search tokens $\{w^*_{i_1}, \ldots, w^*_{i_q}\}$ to their matched documents $\{\mathsf{EDB}(w^*_{i_1}), \ldots, \mathsf{EDB}(w^*_{i_q})\}$, which corresponds to the definition of the L2 leakage profile given above. Actually, if all keywords are queried then the leakage profile L1 collapse to L2. Hence, a passive attack on L2-SSE schemes can be performed on L1-SSE schemes if all keywords have been queried.

4 ATTACKS

Our attacks aim at recovering information on encrypted documents from the knowledge of EDB stored on the server. Hence, the attacker can be a curious server or a malicious person who is able to access the server. These attacks are completely passive; the only assumption made here is that we know a (small) sample $\mathcal{S}$ of the plaintext documents. We emphasize that we do not know any pair of cipher/plain documents. We stress that this knowledge of a sample $\mathcal{S}$ is in practice a realistic assumption: for instance, data sets of mails might contain items that have been transferred outside the scope of the SSE scheme. We can also imagine a user having a part of its data on a server and he decides to encrypt all of its data using a SSE scheme. When the user uploads the encrypted database, the server has the knowledge of both the old plain data and the encrypted database. With these scenarios, we represent the known sample $\mathcal{S}$ by choosing randomly plaintext documents from DB.

Model. 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 performance of this association step heavily depends on the statistic capacity to give unique results over the data set. Assume we are in the case of a data set of books and there is one known best-seller in the data set. An attacker can try to find its identifier in the encrypted database by checking, for example, if there is an unique identifier sharing the length of the known best-seller (L4-SSE schemes) or if there is an unique identifier sharing the same number of distinct keywords (L3-SSE schemes).

In the second step, the keywords of the plaintexts are paired with their tokens. Of course, under L4 and L3 leakage profiles, which preserve the order of keywords in EDB, this pairing is completely straightforward. Finally, correspondences between keywords and tokens obtained from $\mathcal{S}$ can be spread back into EDB, thus recovering partially or totally the content of the encrypted documents. This actually has a devastating effect, giving to the server a massive knowledge of DB, as shown in Section 5.

4.1 Mask Attack on L4-SSE

In order to capture keywords number, order and occurrence counts, 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. For example, if $d_i = (\text{to}, \text{be}, \text{or}, \text{not}, \text{to}, \text{be})$, then $\mathsf{mask}(d_i) = \mathsf{mask}(\mathsf{id}_i) = (1,2,3,4,1,2)$.

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. Hopefully, only one mask of EDB is matching the mask of d, leading to a correct association. In practice, this is almost always the case (see Section 5). The entire process is summarized in Algorithm 1.

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**
  $A_d = \{i \mid \ell_i = \#d, \mathsf{mask}(\mathsf{id}_i) = \mathsf{mask}(d)\}$;
return $\mathsf{W}^*_{\mathsf{rec}} = \{\mathsf{W}^*_i \mid \#A_{d_i} = 1\}$

Algorithm 1: Mask Attack.

4.2 Co-Mask Attack on L3-SSE

Under L3 leakage the Mask Attack does not apply anymore as the mask of a document d boils down to the sequence $(1, \ldots, \#[d])$.

Therefore 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. We recall that $\mathsf{Pos}(w, d)$ is the position of keyword w in document $[d]$ and define:

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

We stress that this quantity can be computed directly from every EDB of profile L3; by abuse of notation this is denoted by $\mathsf{comask}(\mathsf{id}_1, \mathsf{id}_2)$.

The general idea of the algorithm is as follows: for each pair in $(d_i, d_j) \in \mathcal{S}^2$, the co-resulting mask of the pair is computed and compared with all co-resulting masks computed from elements of EDB which have length $\#[d_i]$ and $\#[d_j]$.

In practice, this kind of exhaustive search would be particularly inefficient. We instead 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:

$$A_t = \left\{ \left(\mathsf{id}_{i_1}, \ldots, \mathsf{id}_{i_t}\right) \text{ such that } \forall s, u \le t,\ \mathsf{comask}(\mathsf{id}_{i_s}, \mathsf{id}_{i_u}) = \mathsf{comask}(d_s, d_u) \right\}.$$

Hence, the initialization of the Co-Mask Attack consists for the adversary to compute A_2 corresponding to the pairs of identifiers sharing the same comask that the first considered pair of plain documents known by the adversary. Then, to compute A_t from A_{t-1} using d_t, we consider for each induced new pair (d_j, d_t) the set $C_{j,t}$ of pairs of identifiers $(\mathsf{id}_{i_j}, \mathsf{id}_{i_t})$ with matching co-resulting masks, such that both id_{i_j} and id_{i_t} are still marked as compatible. From the $C_{j,t}$'s, it is easy to remove all inconsistent t-tuples from A_t, i.e. for each j, those having positions j and t not in $C_{j,t}$. When t reaches $\#\mathcal{S}$, the whole search space has been explored: each component $A_{\#\mathcal{S}}[k]$ composed of only one element gives the correct association $A_{\#\mathcal{S}}[k] = \mathsf{id}_k$.

It is worth noting that in practice A_2 is almost always reduced to one element, and so is $A_{\#\mathcal{S}}$. In any case, very few identifiers would remain possible for a given document in $A_{\#\mathcal{S}}$. The Co-Mask Attack is summarized in Algorithm 2. We stress that this attack could be extended to higher order intersections. In practice, only considering pairs already gives outstanding results, as shown in Section 5.

4.3 PowerSet Attack on L2-SSE

As the order of keywords is not preserved anymore under L2 leakage, the co-resulting mask used in the Co-Mask Attack cannot be computed. Worse, even if a document is correctly associated to its identifier, inferring the correct association between each keyword and its token is still a challenge. The PowerSet Attack addresses both issues.

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

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

Algorithm 2: Co-Mask Attack.

Associating Documents and Identifiers. An L2 leakage still allows to determine which keywords are shared between two documents. To associate documents of $\mathcal{S}$ to their identifiers, it is therefore tempting to run the Co-Mask Attack where the co-resulting mask of a pair of documents is replaced by the cardinal of their intersection. Unfortunately this is not sufficient, since in practice many pairs of identifiers of EDB share the same number of tokens.

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, i.e.

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

We stress that this sequence can be computed directly from every EDB of profile L2; by abuse of notation this will be denoted by $\mathsf{PowerSet}^t_h(\mathsf{id}_1, \ldots, \mathsf{id}_t)$.

Example. Let $d_1 = (w_1, w_2, w_3)$, $d_2 = (w_2, w_3)$ and $d_3 = (w_1, w_4)$ be three documents. Then, the PowerSet of order 2 of these three documents is $\mathsf{PowerSet}^3_2(d_1, d_2, d_3) = (\#(\mathsf{W}_1 \cap \mathsf{W}_2), \#(\mathsf{W}_1 \cap \mathsf{W}_3), \#(\mathsf{W}_2 \cap \mathsf{W}_3))$.

The algorithm strives to exploit all available information on $\mathcal{S}$, i.e. finding sequences of identifiers such

that cardinals of all intersections of all possible subsets equal cardinals of those computed on $\mathcal{S}$. As this is a huge search space, it must be explored with care. Therefore, we 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.

Hence, the initialization of the PowerSet Attack consists for the adversary to compute A_2 corresponding to the pairs of identifiers sharing the same number of distinct keywords. Then, computing A_t starting from A_{t-1} and candidate identifiers for d_t requires to reduce the size of A_t as fast as possible. This is done by considering subset intersections of increasing order, thus squeezing A_t as the combinatorics grow. Let $A_t^{(h)}$ be the set of compatible t-tuples with all power sets of order up to h:

$$A_t^{(h)} = \big\{ \big(\mathsf{id}_{i_1},\ldots,\mathsf{id}_{i_t}\big) \text{ such that } \forall s \le h,$$
$$\mathsf{PowerSet}_s^t(d_1,\ldots,d_t) = \mathsf{PowerSet}_s^t(\mathsf{id}_{i_1},\ldots,\mathsf{id}_{i_t}) \big\}.$$

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

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

Input: $\mathcal{S}_t = \big(d_1,\ldots,d_t\big), A_t^{(h)}$
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 $1 \le j_1 < \cdots < j_h < t$ **do**
  $C_{j,t} = \{ \big((\mathsf{id}_{i_j}),\mathsf{id}_{i_t}\big)$ such that $\mathsf{id}_t \in B_t[t]$ $(\mathsf{id}_{i_j}) \in B_t[j]$ and $\#\big(\mathsf{id}_{i_t} \cap (\mathsf{id}_{i_j})\big) = \#\big(d_t \cap (d_j)\big)\}$;
  $B_t = \big\{ b \in B_t \mid \big((b[j]),b[t]\big) \in C_{j,t} \big\}$;
  // Keep consistent t-tuples
  if $\#B_t = 1$ **then** break;
return $A_t^{(h+1)} = B_t$

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

We stress that, by induction, only subsets containing d_t have to be considered. Algorithm 4 summarizes the first phase of the PowerSet Attack.

In practice, computing A_2 is the most costly part of Algorithm 4, as the result is sufficiently small so that adding new documents becomes negligible.

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
$A_2 = \big\{ \big(\mathsf{id}_{i_1},\mathsf{id}_{i_2}\big)$ such that $\#\mathsf{id}_{i_1} = \#\{d_1\}$, $\#\mathsf{id}_{i_2} = \#\{d_2\}$ and $\mathsf{PowerSet}_2^t(\mathsf{id}_{i_1},\mathsf{id}_{i_2}) = \mathsf{PowerSet}_2^t(d_1,d_2)\big\}$;
// 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;
return $\mathcal{S}_0 = \big\{ d_t \mid \#A_{\#\mathcal{S}}[t] = 1 \big\}$

Algorithm 4: PowerSet Attack: documents-identifiers association.

Moreover, experiments produced on chosen data sets (*Commons*, *Enron*, *Gutenberg* and *Lucene*) show that A_t is reduced to one element as soon as $t \ge 4$.

Associating Keywords and Tokens. The previous phase associates each document of $\mathcal{S}_0$ with a set of tokens. Since token ordering is not preserved under L2 leakage, finding the correct keyword-token associations remains non-trivial.

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* $\mathsf{inv}_{\ge}(\mathcal{S}_0)$.

Consider first the keyword w_i having the most identifiers, and assume that no following keyword has the same associated identifiers. Hence the intersection of the sets of tokens associated to w_i gives a unique match w_i^*. Now, if the second line w_j of $\mathsf{inv}_{\ge}(\mathcal{S}_0)$ is also unique, we distinguish two cases: either the intersection of the sets of tokens associated to w_j gives a unique match w_j^*; or, when identifiers are also associated to the previous keyword w_i, we obtain two tokens. Knowing w_i^* from the first association, we easily deduce the token w_j^* associated to w_j.

Example. Let $\mathcal{S}_0 = \{d_1,d_2,d_3\}$ be a set of three documents $d_1 = (w_1,w_2,w_3)$, $d_2 = (w_3,w_2)$ and $d_3 = (w_1,w_2)$. Inverted indexes $\mathsf{inv}(\mathcal{S}_0)$ and $\mathsf{inv}_{\ge}(\mathcal{S}_0)$ are:

$\mathsf{inv}(\mathcal{S}_0)$	
w_1	id_1 id_3
w_2	id_1 id_2 id_3
w_3	id_1 id_2

$\mathsf{inv}_{\ge}(\mathcal{S}_0)$	
w_2	id_1 id_2 id_3
w_3	id_1 id_2
w_1	id_1 id_3

Consider the first line of $\mathsf{inv}_{\geq}(\mathcal{S}_0)$. We know that only w_2 is in d_1, d_2 and d_3. Hence $\mathsf{W}_1^* \cap \mathsf{W}_2^* \cap \mathsf{W}_3^* = \{w_2^*\}$. Now, consider the second keyword of $\mathsf{inv}_{\geq}(\mathcal{S}_0)$ i.e. w_3. This keyword is in d_1 and d_2, but w_2 too. So $\mathsf{W}_1^* \cap \mathsf{W}_2^* = \{w_2^*, w_3^*\}$, but we already know that w_2^* is the token of w_2, hence the token of w_3 is w_3^*.

Unfortunately, several keywords may be associated to the same identifiers. In this case, they are completely indistinguishable and we ignore them when they appear in the following intersections. This process is given in Algorithm 5.

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

Algorithm 5: PowerSet Attack: keywords-tokens association.

4.4 Elements of Complexity

Deriving complexity bounds for our attacks depend on statistical properties of the targeted data set. We nevertheless give some elements allowing to compare the impact of the leakage profiles.

The most relevant data for our attacks is the maximum number of identifiers to consider for a document of a given length. For each leakage profile, we have $\mathsf{M}_{\mathsf{L23}} = \max_{d \in \mathsf{DB}} \#\{\mathsf{id} \mid \#\mathsf{id} = \#[d]\}$ and $\mathsf{M}_{\mathsf{L4}} = \max_{d \in \mathsf{DB}} \#\{\mathsf{id} \mid \#\mathsf{id} = \#d\}$.

Measurements on our data sets (see Section 5) show that $\sqrt{\#\mathsf{DB}}$ is a good approximation of these values.

Mask Attack. For each known document d of $\mathcal{S}$, the Mask Attack computes masks for all candidates of d, i.e. M_{L4} masks computation for each document d. Hence the total complexity for the Mask Attack is $O(\#\mathcal{S} \cdot \mathsf{M}_{\mathsf{L4}})$ mask computations.

Co-Mask Attack. The Co-Mask Attack starts with the construction of A_2, i.e. the set of all identifiers pairs of the encrypted database sharing the same comask of the two chosen known documents of $\mathcal{S}$. Hence, constructing A_2 costs $\mathsf{M}_{\mathsf{L23}}^2$ applications of comask since we check all candidate pairs for the initial comask. We heuristically expect the sets A_t to decrease as t grows. Indeed, if $\#A_2 \leq \mathsf{M}_{\mathsf{L23}}$, each association of d_t starts from a smaller set A_{t-1} and imposes greater constraints, thus costing at most $\#A_2 \cdot \mathsf{M}_{\mathsf{L23}}$. In our experiments with chosen data sets, A_2 is almost always reduced to one element. Since, we check the comask for all candidates of each document d of $\mathcal{S}$ after the initialization, we conjecture a total complexity of $O(\mathsf{M}_{\mathsf{L23}}^2 + \#\mathcal{S} \cdot \mathsf{M}_{\mathsf{L23}})$ co-mask computations.

PowerSet Attack. The analysis is much more complex. As the Co-Mask Attack, the PowerSet Attack starts with the construction of A_2, i.e. the set of all identifiers pairs sharing the same number of keywords of the two chosen known documents of $\mathcal{S}$. Hence, constructing A_2 costs $\mathsf{M}_{\mathsf{L23}}^2$ intersections cardinals computations since we check all candidate pairs for the initial cardinal intersection. Heuristically, the first pair considered drastically reduces the number $\#A_2$ of candidates, and the same reasoning as above leads to a conjectured complexity of $O(\mathsf{M}_{\mathsf{L23}}^2 + \#A_2 \cdot \#\mathcal{S} \cdot \mathsf{M}_{\mathsf{L23}})$ intersections cardinals computations.

Table 1: Characteristics of used data sets.

Data sets	Content	#DB	#W	N
Commons	mailing list	28,997	230,893	3,910,562
Enron	emails	490,369	643,818	47,301,160
Gutenberg	books	21,602	2,853,955	91,261,811
Lucene	mailing list	58,884	394,481	7,952,794

5 EXPERIMENTAL RESULTS

Real-World Data Sets. We implemented and ran the attacks presented in Section 4 on four different real-world data sets to evaluate their practical efficiency.

The first data set is the email data set from the *Enron* corporation, available online[¶]. Islam et al. (Islam et al., 2012) and Cash et al. (Cash et al., 2015) consider emails from each employee's sent mail. Here, we choose to took all 490,369 emails of the data sets, including mails sent from the outside of *Enron*. The second and third data sets are mailing lists from the *Apache* foundation, namely *Apache Commons*[‖] and *Apache Lucene* which is used too in (Cash et al., 2015; Islam et al., 2012). The last data set is the *Project Gutenberg*[**]. We summarize characteristics of used data sets in Tab. 1.

One email message, one article or one book is considered as one document. For each document, stopwords have been removed. Moreover, we use the stan-

[¶] cs.cmu.edu/~./enron/
[‖] mail-archives.apache.org/mod_mbox/commons-user/
[**] gutenberg.org/wiki/Main_Page

dard Porter stemming algorithm (Porter, 1980) to find the root of each word of data set documents. We stress that all processing steps on keywords have been done considering the result given by the Porter's algorithm.

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 data set; this allows us to perform a fairer comparison between data sets than the usual per-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 data set. In particular, we measured the rate of documents of the data set whose keywords are recovered at 70%, 80%, 90% and 100%.

Experimental Results on Lucene. We expose here the results of our attacks on the *Lucene* data set. All timings are measured on a Core i7 using 16 Gb RAM.

Our attacks have a huge impact. If the server only knows 1% of the *Lucene* data set, the Mask attack (resp. the Co-Mask attack) can recover 99% of keywords present in this sample in 72 seconds (resp. 284 seconds) whereas the PowerSet Attack can even so recovers 21% of keywords in 489 seconds. The impact on the knowledge of the protected data is illustrated with graphs in Fig. 1. For the Mask and the Co-Mask attacks, the recovering of the 99% of keywords present in the 1% of the data set allows us to recover 50% of the protected data at 90%. For the PowerSet Attack, the recovering of the 21% of keywords present in the 1% of the data set allows to recover 25% of the protected data at 90%. Details are presented in Tab. 2. We precise that the impact of our attacks is the same on the others chosen data sets (*Commons*, *Enron* and *Gutenberg*)[††].

Mask Attack. Over 98% of documents have a unique mask in *Lucene* data set. This translates into over 99% keyword-token association rate over the set $\mathcal{S}$ in all cases. Moreover, knowing only 1% of the data set already allows the server to recover 70% of the keywords close to all documents; and 3,146 of them are completely recovered.

Co-Mask Attack. Experiments show that despite the loss of the frequency information, it remains as effective as the Mask Attack.

PowerSet Attack. It suffers widely from the loss of keyword order. Hence, while the documents-identifiers association performs equally well, the exact association between keywords and tokens plateau around 20%. Still, the knowledge of 1% of the data set already allows to recover 80% of the keywords of more than 70% of the database documents.

Practical Impact. As noted in (Cash et al., 2015), 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.

[††]Results for *Commons*, *Enron* and *Gutenberg* are available online: https://iacr.org/2017/046.pdf

6 GAP BETWEEN L2- AND L1-SSE

We discuss the gap for an adversary to recover client queries between L2- and L1-SSE schemes.

IKK Attack. (Islam et al., 2012) present a passive query recovery attack on SSE schemes. It requires access to a co-occurrence matrix C_{W} which represents the probability for two keywords to appear in a randomly chosen document. The attack also requires the observation of queries issued by the client and the responses provided by the server. The adversary is then able to compute for each pair of search tokens, the number of documents which match for both. Associating keywords to search tokens boils down to finding the minimum of the function $F(i_1,\ldots,i_q) =$

$$\sum_{1\leq s,t\leq q}\left|\frac{\#\big(\mathsf{EDB}(w_s^*)\cap\mathsf{EDB}(w_t^*)\big)}{n}-C_{\mathsf{W}}\big(w_{i_s},w_{i_t}\big)\right|^2,$$

for observed search tokens $(w_1^*,\ldots,w_q^*)$. Since no assumption are made about the amount of leakage obtained from the SSE scheme, we can classify the IKK attack as an L1 attack with auxiliary information in the form of this co-occurrence matrix C_{W}. To the best of our knowledge this is the most generic attack on SSE schemes. Islam et al. justify the access to a co-occurrence matrix by implying that it could be computed from a data set *similar* to the one targeted by the attack. The cost of building C_{W} could then be amortized over several data sets. In practice, (Cash et al., 2015) show that any kind of success with this attack requires C_{W} to have been computed directly from the plaintext data set DB. Another constraint is that for the attack to be practical C_{W} cannot be built over the full dictionary; we must assume that all the search tokens are associated to a keyword represented in C_{W}. Following Section 3, we can relax the requirements by considering this attack on an L2 scheme. In this setting the adversary computes the response set intersections directly from EDB without the need for search tokens. Complexity-wise, the IKK attack is costly as minimizing the objective function F requires the use of simulated annealing (Islam et al., 2012).

Count Attack. The Count attack from Cash et al. in (Cash et al., 2015) also aims at passively re-

Table 2: Rate of recovered keywords and of 80%, 90% and 100% recovered documents of *Lucene* data set.

%DB known	Mask Attack				Co-Mask Attack				PowerSet Attack			
	Rate	# 80%	# 90%	# 100%	Rate	# 80%	# 90%	# 100%	Rate	# 80%	# 90%	# 100%
1	0.99	0.88	0.48	0.05	0.99	0.87	0.48	0.05	0.21	0.71	0.24	0.14
2	0.99	0.93	0.62	0.10	0.99	0.92	0.61	0.10	0.20	0.81	0.36	0.03
3	0.99	0.95	0.68	0.13	0.99	0.94	0.68	0.13	0.19	0.85	0.44	0.04
4	0.99	0.95	0.72	0.16	0.99	0.95	0.72	0.16	0.19	0.87	0.49	0.04
5	0.99	0.96	0.75	0.19	0.99	0.96	0.75	0.19	0.19	0.89	0.53	0.06
10	0.99	0.98	0.83	0.30	0.99	0.98	0.83	0.30	0.16	0.93	0.63	0.09
20	0.99	0.99	0.89	0.46	0.99	0.99	0.89	0.46	0.16	0.95	0.71	0.15
30	0.99	0.99	0.92	0.57	0.99	0.99	0.92	0.57	0.16	0.96	0.75	0.19
40	0.99	0.99	0.94	0.66	0.99	0.99	0.94	0.66	0.16	0.97	0.77	0.22
50	0.99	0.99	0.96	0.74	0.99	0.99	0.96	0.74	0.16	0.97	0.79	0.25

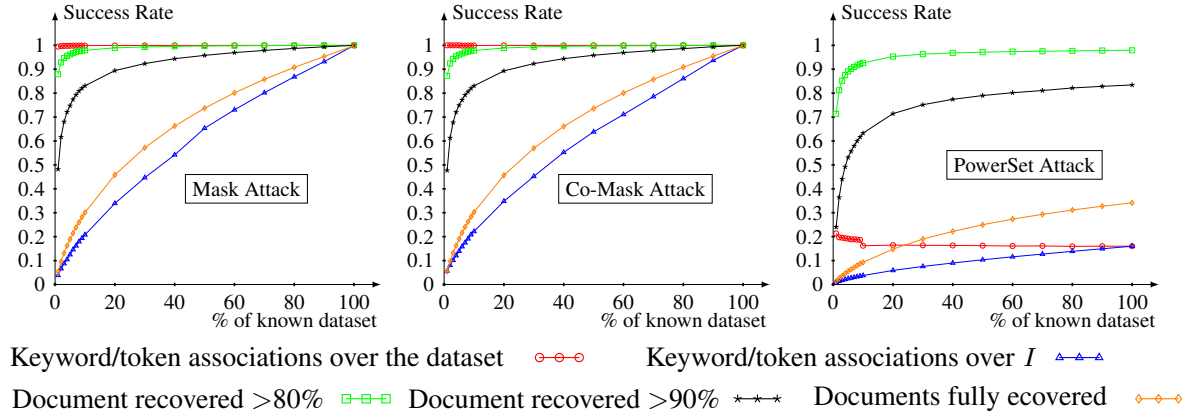

Figure 1: Efficiency of our attacks on *Lucene* data set depending on the knowledge rate of the server.

covering queries with the help of queries and a C_{W}. However on top of that, it requires to have access for each keyword to the number of plaintext documents that contain it. The adversary is then able to match search tokens to a set of candidate keywords. Wrong candidates are then eliminated using C_{W}.

Count Attack assumes that the adversary has access to the pairs $\big(w_i, \#\mathsf{DB}(w_i)\big)_{1\leqslant i\leqslant n}$ from which he can compute the set $\big\{\#\mathsf{DB}(w_1),\ldots,\#\mathsf{DB}(w_n)\big\} = \big\{\#\mathsf{EDB}(w^*_{i_1}),\ldots,\#\mathsf{EDB}(w^*_{i_n})\big\}$.

The use of a co-occurrence matrix means that the Count Attack shares properties with the IKK Attack: namely we assume that the observed search queries correspond to keywords in C_{W} and we do not need search queries anymore if we attack an L2 scheme. Complexity-wise, the Count Attack is orders of magnitude faster than the IKK Attack since we leverage the extraneous auxiliary information to avoid doing any numerical optimization step.

Shadow Nemesis Attack. The Shadow Nemesis Attack from Pouliot and Wright in (Pouliot and Wright, 2016) presents also a passive query recovery attack on SSE schemes. First, it uses a training set to build an approximate co-occurrence matrix C_{W}. Then, it build a second co-occurrence matrix C'_{W} from the encrypted database with keywords that has been queried. From C_{W} (resp. C'_{W}), they construct the weighted graph G (resp. H). Pouliot and Wright compare these two co-occurrence matrix by reducing them to the combinatorial optimization problem of weighted graph matching. The problem is to find the permutation X that relabels the nodes in H so that the permuted graph most closely resembles G. If A_G and A_H are respectively the adjacency matrices of G and H and using the Euclidean distance denoted $\|\cdot\|_2$, then the goal is to find X such that minimizes: $\|A_G - X\cdot A_H\cdot X^T\|_2$.

Comparison between attacks. We emphasize that PowerSet and Shadow Nemesis attacks target L2-SSE while IKK and Count attacks target L1-SSE. We compare the PowerSet Attack to the previous attacks with same settings considering the recovered rate of 150 keywords uniformly chosen from the 1 500 most common keywords. The co-occurrence matrix used by IKK, Count and Shadow Nemesis attacks is approached via the sample $\mathcal{S}$ known by the adversary. Figure 2 reveals the gap that exists between L2-SSE schemes and L1-SSE schemes which reveal less information. Indeed, in spite of auxiliary information and information from queries used by IKK and Count at-

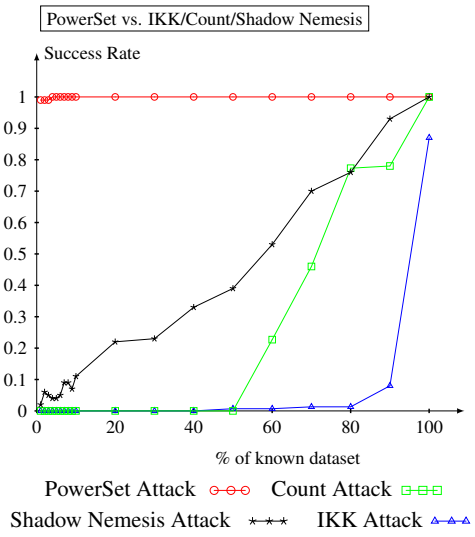

Figure 2: Most commons keywords recovery rates. *Lucene* data set, 1 500 keywords, 150 chosen uniformly.

tacks, Figure 2 shows that L1-SSE schemes are more resistant to recover client queries. If the adversary only knows 5% of the data set, our attack can recover 100% of the 150 keywords while the Shadows Nemesis Attack recovers only 5% of them and whereas IKK and Count attacks need to know more than 60% of data set to recover keywords.

7 COUNTERMEASURES

The countermeasure for the PowerSet Attack is generic but computationally costly while countermeasure for the Co-Mask and the Mask attacks are specific and computationally efficient.

Countermeasure for the PowerSet Attack. Since the PowerSet Attack uses information on number of shared keywords between documents, the idea is to modify plaintext documents such that there exists at least $\alpha - 1$ documents having the same keywords for each document. In this way, when the PowerSet Attack is performed, there is at least α different tuples of identifiers corresponding to the power set computed from the sample $\mathcal{S}$. Hence, the adversary cannot deduce the correct association between plaintext documents that she knows and their identifiers. To do that, we are inspired by (Islam et al., 2012). We consider the database DB as a binary matrix M of size $m \cdot n$, where $M_{i,j} = 1$ if $w_i \in d_j$ and $M_{i,j} = 0$ if $w_i \notin d_j$. Since the PowerSet Attack is performed only on the encrypted database and does not require any query, we modify the matrix M such that for each column there exist $\alpha - 1$ similar columns. We authorize only

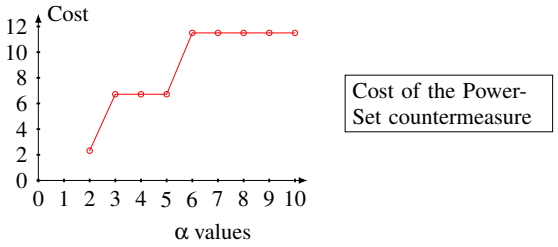

Figure 3: Cost of the PowerSet countermeasure for different α values on a sample of 1 000 documents from *Commons*.

false-positive that is modify a 0 to a 1. Then our aim is to minimize the number of false-positive, so we use an algorithm of agglomerative hierarchical clustering with average distance and the *cosine distance* as distance measure (Berkhin, 2006). When each cluster have at least α documents, if there is at least one column having 1 to its j-th then we put 1 into the j-th row of all columns of the cluster. This countermeasure preserves the size of the original database at the cost of false-positives. As in (Islam et al., 2012), we define cost as the ratio of number of documents returned by the new encrypted database (denoted by q) to the number of documents returned by the old encrypted database (denoted by p). That is $cost = (q - p)/p$. Fig. 3 shows the evolution of the cost of the countermeasure for different α values when the presented countermeasure is performed on a sample of 1 000 documents from the data set *Commons*. The stages of the cost are explained by the number of documents in each cluster. In fact, when $\alpha = 3$, each cluster have already 5 documents, hence clusters do not change when α changes from 3 to 5.

Countermeasure for the Co-Mask Attack. The Co-Mask Attack targets L3-SSE. These schemes leak the order of keywords first appearance. Assume we want to keep the relative order of keywords in documents to allow the scheme to sort replies from the server in function of the queried keyword position. As the countermeasure for the PowerSet Attack, we authorize false positives. The idea is to add a keyword of W which is not in the initial document. Its position is randomly chosen when the client builds the encrypted database. Hence, an adversary knowing this countermeasure and a sample $\mathcal{S}$ of plain documents has on one hand a low probability to choose the same keyword, and on the other hand has a low probability to choose the same position. This countermeasure decreases the chance to have a match between comask computed by the adversary from $\mathcal{S}$ and those which are computed directly from the encrypted database. Moreover, we only add one false positive by document.

Countermeasure for the Mask Attack. The Mask

Attack targets L4-SSE schemes. Assume we want to keep information on occurrence and order of keywords for the same reason as above. Again, we authorize false positives. Hence we can add a random keyword at a random position in each document. In this way, the mask of the original document does not correspond to those of the new document. Moreover, if the adversary tries to find the correct identifier of a document in the encrypted database, it has a low probability to find the added keyword and its position. A possible alternative to not add false positive is to choose the added keyword among those of the original document. This increases the chance for the adversary to guess the added keyword.

8 CONCLUSION

Prior work (Zhang et al., 2016) taught us that SSE schemes have no hope of being secure in a setting where the adversary can inject chosen files. Additionally, (Cash et al., 2015; Islam et al., 2012; Pouliot and Wright, 2016) have shown that passive observations of search tokens reveal the underlying searched keyword when the data set is fully known. This paper focuses on passive attacks of L4, L3 and L2 schemes currently used as commercially solutions, e.g. CipherCloud. The most glaring conclusion is that our attacks are devastating and have a real impact on the protected data in the cloud: regardless of the leakage profile, knowing a mere 1% of the document sets translates into over 90% of documents whose content is revealed over 70%. Moreover, having same knowledge from the data set, we show that we recover same rate of keywords whether it is with L4- or with L3-SSE schemes. We show too that the gap of security that exists between L2- and L1-SSE schemes is important since L1 attacks need to know a large amount of information to recover frequent keywords contrary to our L2 attack. Our results give a better understanding of the practical security of SSE schemes and hopefully will help practitioners make more secure SSE schemes. Future work may deal with countermeasures in depth and with the study of the degradation from L1 to L2 in the presence of queries.

ACKNOWLEDGEMENTS

This research was conducted with the support of the FEDER program of 2014-2020 and the region council of Auvergne-Rhône-Alpes.

REFERENCES

Berkhin, P. (2006). A Survey of Clustering Data Mining Techniques.

Cash, D., Grubbs, P., Perry, J., and Ristenpart, T. (2015). Leakage-Abuse Attacks Against Searchable Encryption. In *CCS 2015*, New York, NY, USA. ACM.

Cash, D., Jaeger, J., Jarecki, S., Jutla, C. S., Krawczyk, H., Rosu, M., and Steiner, M. (2014). Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*.

Cash, D., Jarecki, S., Jutla, C. S., Krawczyk, H., Rosu, M., and Steiner, M. (2013). Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *CRYPTO 2013*.

Curtmola, R., Garay, J. A., Kamara, S., and Ostrovsky, R. (2006). Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS 2006*.

Faber, S., Jarecki, S., Krawczyk, H., Nguyen, Q., Rosu, M., and Steiner, M. (2015). Rich Queries on Encrypted Data: Beyond Exact Matches. In *ESORICS 2015*.

Goldreich, O. (1998). *Secure Multi-party Computation*. Working Draft.

He, W., Akhawe, D., Jain, S., Shi, E., and Song, D. (2014). ShadowCrypt: Encrypted Web Applications for Everyone. In *CCS 2014*.

Islam, M. S., Kuzu, M., and Kantarcioglu, M. (2012). Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *NDSS 2012*.

Kamara, S., Papamanthou, C., and Roeder, T. (2012). Dynamic Searchable Symmetric Encryption. In *CCS 2012*.

Lau, B., Chung, S., Song, C., Jang, Y., Lee, W., and Boldyreva, A. (2014). Mimesis Aegis: A Mimicry Privacy Shield–A System's Approach to Data Privacy on Public Cloud. In *USENIX Security 2014*.

Porter, M. F. (1980). An algorithm for suffix striping. *Program*.

Pouliot, D. and Wright, C. V. (2016). The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption. In *CCS 2016*.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical Techniques for Searches on Encrypted Data. In *SP 2000*. IEEE Computer Society.

Zhang, Y., Katz, J., and Papamanthou, C. (2016). All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. Cryptology ePrint Archive, Report 2016/172.