Practical Passive Leakage-Abuse Attacks
Against Symmetric Searchable Encryption

Matthieu Giraud¹, Alexandre Anzala-Yamajako², Olivier Bernard², and Pascal Lafourcade¹

¹Université Clermont Auvergne, BP 10448, F-63000, Clermont-Ferrand, France.
{firstname.lastname}@uca.fr

²Thales 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\textsuperscript{a} and Skyhigh Networks\textsuperscript{b}, L3 schemes such Bitglass\textsuperscript{c} or L2 schemes as ShadowCrypt (He et al., 2014) and Mime
tesis (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\textsuperscript{d}. 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-
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 (…), lists are denoted by square brackets […] and sets by braces {…}. The number of elements of a set $E$ (resp. list or sequence) is written $\# E$.

Documents and keywords. Let $W = \{w_1, \ldots, w_m\}$ be a dictionary composed of $m$ distinct keywords and $DB = \{d_1, \ldots, d_n\}$ a set of $n$ documents made of keywords from $W$. Each document $d_i$ is a sequence of length $\ell_i$, formally $d_i = (w_{i1}, \ldots, w_{i\ell_i}) \in W^{\ell_i}$. DB is called the data set. We denote by $W_i$ the set of distinct keywords of the document $d_i$, i.e. $W_i = \{ \{w_i\}\}$.

The same objects are described server-side by introducing the star superscript. Hence, $W^* = \{w^*_1, \ldots, w^*_m\}$ denotes the set of search tokens associated to the keywords of $W$. Similarly, $DB^* = \{d^*_1, \ldots, d^*_n\}$ 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 $d_i \in DB$ contains the keyword $w$ is denoted by $DB(w)$. $N$ denotes the number of pairs $(d, w)$ where $d \in DB$ and $w \in d$, i.e. $N = \# \{(d, w) \mid d \in DB, w \in d\}$. 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 $d^*_i \in DB^*$ associated to the search token $w^*$ is written $EDB(w^*)$. We stress that this information is not accessible directly from $w^*$ and $DB^*$, we need the extra protected metadata structure $EDB$. Moreover, $Pos(w, d)$ 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 $(E(\cdot), D(\cdot))$ where $E(\cdot)$ denotes the encryption algorithm and $D(\cdot)$ denotes the decryption algorithm, we define a static SSE scheme of security parameter $\lambda$ as a quartet of polynomial-time algorithms $\Pi = (Gen, \Pi.Setup, SearchClient, SearchServer)$ by:

$$(K, k) \leftarrow Gen(\lambda)$$ is a probabilistic algorithm run by the client. It takes as input a security parameter $\lambda$, and outputs two symmetric secret keys $K$ and $k$ which are both kept securely by the client.

$$(EDB, DB^*) \leftarrow \Pi.Setup(K, k, DB, E(\cdot))$$ 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 $E(\cdot)$, and outputs both the protected metadata $EDB$ and the encrypted documents $DB^* = (E_k(d_1), \ldots, E_k(d_n))$.

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

$\text{EDB}(w^*) \leftarrow \text{SearchServer}(\text{EDB}, w^*)$ is a deterministic algorithm run by the server to answer a client-query. It takes as input the protected meta-data EDB and the client-generated search token $w^*$ and outputs $\text{EDB}(w^*)$: 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 $L$ 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 $L$-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 $L$. The $L$-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).

#### 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. 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 $L$ is of class L4 if and only if $L(\text{EDB}) = \{w_{i1}^*, \ldots, w_{iL}^*\}_{1 \leq i \leq n}$.

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

<table>
<thead>
<tr>
<th>$W$</th>
<th>$W^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>14</td>
</tr>
<tr>
<td>call</td>
<td>76</td>
</tr>
<tr>
<td>i</td>
<td>33</td>
</tr>
<tr>
<td>if</td>
<td>17</td>
</tr>
<tr>
<td>me</td>
<td>25</td>
</tr>
<tr>
<td>you</td>
<td>10</td>
</tr>
<tr>
<td>possible</td>
<td>35</td>
</tr>
<tr>
<td>soon</td>
<td>78</td>
</tr>
</tbody>
</table>

Under L4 leakage, $\text{EDB}$ reveals to the server $(76, 25, 14, 78, 14, 35) \rightarrow id_1$ and $(33, 76, 10, 11, 35) \rightarrow id_2$. 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.

#### 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 $L$ is of class L3 if and only if $L(\text{EDB}) = \{L_{\text{EDB}}(w_i)\}_{1 \leq i \leq n}$, where $L_{\text{EDB}}(w_i) = \{(w_{i1}^*, \ldots, w_{iL}^*)\}$.

**Example.** Resuming the running example, the information revealed by an L3-SSE scheme about $d_1$ and $d_2$ is: $(76, 25, 14, 78, 35) \rightarrow id_1$ and $(33, 76, 10, 11, 35) \rightarrow id_2$. The server does not know anymore that the token 14 is associated twice to $id_1$.

#### L2 Leakage Profile

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 $L$ is of class L2 if and only if $L(\text{EDB}) = \{w_i^*\}_{1 \leq i \leq n}$.

**Example.** Resuming the running example, an L2-SSE scheme reveals about $d_1$ and $d_2$: $(14, 25, 35, 76, 78) \rightarrow id_1$ and $(10, 11, 33, 35, 76) \rightarrow id_2$. We stress that the token order is not preserved in $\text{EDB}$: we arbitrarily sort the token in ascending order, thus the server does not know their initial order.

#### L1 Leakage Profile

With no initial search, 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 $L(\text{EDB}) = \{N\}$.

**Example.** Resuming the running example, the information revealed by an L1-SSE scheme looks like:

<table>
<thead>
<tr>
<th>$w_i^*$</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\gamma$</th>
<th>$\delta$</th>
<th>$\epsilon$</th>
<th>$\zeta$</th>
<th>$\eta$</th>
<th>$\theta$</th>
<th>$\iota$</th>
<th>$\kappa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>( w^* )</th>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>( \gamma )</th>
<th>( \delta )</th>
<th>( \epsilon )</th>
<th>( \iota )</th>
<th>( \iota^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Id.} )</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
</tr>
</tbody>
</table>

Hence, the server learns that documents identified by \( \text{id}_1 \) and \( \text{id}_2 \) share the same keyword of token 35; keywords of tokens 14 and 35 are both in the document database, but it is informative to reflect upon the effect of queries on the amount of information revealed.

**Effect of Queries on the \( L_1 \) 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 as-

**Input:** EDB, \( S \subseteq \text{DB} \)

**Output:** Set of tokens \( W^*_\text{rec} \subseteq W^* \) associated to their keyword in \( W \)

| \( A_d = \{ i \mid \ell_i = \#d, \text{mask}(\text{id}_i) = \text{mask}(d) \} \) |
| \( \text{return } W^*_\text{rec} = \{ W^*_i \mid \#A_d = 1 \} \) |

**Algorithm 1:** Mask Attack.

**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. \( \text{id}_i \)), denoted by \( \text{mask}(d_i) \) (resp. \( \text{mask}(\text{id}_i) \)), 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 \( \text{mask}(d_i) = \text{mask}(\text{id}_i) = (1, 2, 3, 4, 1, 2) \).

The idea of the attack is intuitive: for each plaintext document \( d \in S \), 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.

**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
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 Pos($w, d$) is the position of keyword $w$ in document $d$ and define:

$$\text{comask}(d_1, d_2) = \left( (\text{Pos}(d_1[i], d_2))_{1 \leq i \leq \#W_1}, (\text{Pos}(d_2[i], d_1))_{1 \leq i \leq \#W_2} \right).$$

We stress that this quantity can be computed directly from every EDB of profile L3; by abuse of notation this is denoted by comask($id_1, id_2$).

The general idea of the algorithm is as follows: for each pair in $(d_1, d_2) \in S^2$, the co-resulting mask of the pair is computed and compared with all co-resulting masks computed from elements of EDB which have length $\#d_1$ and $\#d_2$.

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 $(d_1, \ldots, d_t) \subseteq S$. More formally:

$$A_t = \{ (id_{i1}, \ldots, id_{it}) \mid \forall s, u \leq t, \text{comask}(id_{is}, id_{it}) = \text{comask}(d_{is}, d_{it}) \}. $$

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 $(d_j, d_k)$ the set $C_{j,k}$, of pairs of identifiers $(id_j, id_k)$, with matching co-resulting masks, such that both $id_j$ and $id_k$ are still marked as compatible. From the $C_{j,k}$, 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 $C_{j,k}$. When $t$ reaches $\#S$, the whole search space has been explored: each component $A_{S}[k]$ composed of only one element gives the correct association $A_{S}[k] = id_k$.

It is worth noting that in practice $A_2$ is almost always reduced to one element, and so is $A_{S}$. In any case, very few identifiers would remain possible for a given document in $A_{S}$. 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.

**Associating Documents and Identifiers.** An L2 leakage still allows to determine which keywords are shared between two documents. To associate documents of $S$ 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 PowerSet$_h(d_1, \ldots, d_t)$, and defined as the sequence of the $\binom{t}{h}$ cardinals of all possible intersections of $h$ elements of the $t$-uple, i.e.

$$\text{PowerSet}_h(d_1, \ldots, d_t) = \left( \# \bigcap_{1 \leq j \leq h} W_{d_j} \right)_{1 \leq h < \leq t}.$$ 

We stress that this sequence can be computed directly from every EDB of profile L2; by abuse of notation this will be denoted by PowerSet$_h(id_1, \ldots, id_t)$.

**Example.** Let $d_1 = (w_1, w_2, w_3)$, $d_2 = (w_2, w_3)$ and $d_3 = (w_1, w_3)$ be three documents. Then, the PowerSet of order 2 of these three documents is PowerSet$_2(d_1, d_2, d_3) = (\#(W_1 \cap W_2), \#(W_1 \cap W_3), \#(W_2 \cap W_3))$.

The algorithm strives to exploit all available information on $S$, i.e. finding sequences of identifiers such
that cardinals of all intersections of all possible subsets equal cardinals of those computed on \( S \). As this is a huge search space, it must be explored with care. Therefore, we iteratively construct a set \( A_t \) containing all \( r \)-tuples of identifiers such that all power sets of order less than \( r \) correspond to the power sets of the corresponding documents in \( (d_1, \ldots, d_t) \in S \). When \( t \) reaches \(#S\), all information on \( S \) has been processed and singleton components of \( A_{#S} \) 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 \( r \)-tuples with all power sets of order up to \( h \):

\[
A_t^{(h)} = \{ (id_1, \ldots, id_h) \text{ such that } \forall s \leq h, \text{PowerSet}^s(d_1, \ldots, d_h) = \text{PowerSet}^s(id_1, \ldots, id_h) \}.
\]

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

\[
A_{t-1} \times \{ \text{id} \mid \#\text{id} = \#d_t \} = A_t^{(1)} \supseteq A_t^{(2)} \supseteq A_t^{(3)} \cdots \supseteq A_t^{(h)} = A_t.
\]

**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.

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

**Associating Keywords and Tokens.** The previous phase associates each document of \( S_0 \) 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 \( S_0 \), denoted by \( \text{inv}(S_0) \), which associates the keywords \( w \in S_0 \) 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 \( \text{inv}_2(S_0) \).

Consider first the keyword \( w_1 \) 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 \( w_1 \) gives a unique match \( w_1^* \). Now, if the second line \( w_1^* \) of \( \text{inv}_2(S_0) \) is also unique, we distinguish two cases: either the intersection of the sets of tokens associated to \( w_1 \) gives a unique match \( w_1^* \); or, when identifiers are also associated to the previous keyword \( w_1 \), we obtain two tokens. Knowing \( w_1^* \) from the first association, we easily deduce the token \( w_2^* \) associated to \( w_2 \).

**Example.** Let \( S_0 = \{ d_1, d_2, d_3 \} \) be a set of three documents \( d_1 = \{ w_1, w_2, w_3 \} \), \( d_2 = \{ w_2, w_3 \} \) and \( d_3 = \{ w_1, w_2 \} \). Inverted indexes \( \text{inv}(S_0) \) and \( \text{inv}_2(S_0) \) are:

<table>
<thead>
<tr>
<th>( \text{inv}(S_0) )</th>
<th>( \text{inv}_2(S_0) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 )</td>
<td>( w_1 )</td>
</tr>
<tr>
<td>( w_1 )</td>
<td>( w_1 )</td>
</tr>
<tr>
<td>( w_2 )</td>
<td>( w_2 )</td>
</tr>
<tr>
<td>( w_2 )</td>
<td>( w_2 )</td>
</tr>
<tr>
<td>( w_3 )</td>
<td>( w_3 )</td>
</tr>
<tr>
<td>( w_3 )</td>
<td>( w_3 )</td>
</tr>
</tbody>
</table>

**Algorithm 4: PowerSet Attack: documents-identifiers association.**

| Input: EDB, \( S = (d_1, \ldots, d_{#S}) \subseteq DB \) |
| Output: Set of documents \( S_0 \subseteq S \) associated to their identifiers in EDB |

// Consider the first pair of documents
\( A_2 = \{ (id_{i_1}, id_{i_2}) \text{ such that } \#id_{i_1} = \#d_1, \#id_{i_2} = \#\{d_2\} \} \) and \( \text{PowerSet}_2(id_{i_1}, id_{i_2}) = \text{PowerSet}_2(d_1, d_2) \);

// Construct \( A_1 \) from \( A_0 \) using \( d_1 \)
for \( t = 3 \) to \#S do

\[
A_t^{(1)} = A_{t-1} \times \{ \text{id} \mid \#\text{id} = \#d_t \};
\]

// Consider intersections of increasing order
\( h \) to reduce \( A_t \)
for \( h = 2 \) to \( t \) do

\[
A_t^{(h)} = \text{Reduce}\left(A_t^{(h-1)}\right);
\]

if \( \#A_t^{(h)} = 1 \) then set \( A_t = A_t^{(h)} \) and break;

return \( S_0 = \{ d_t \mid \#A_{#S}[t] = 1 \} \)
Consider the first line of \( \text{inv} \geq (S_0) \). We know that only \( w_2 \) is in \( d_1, d_2 \) and \( d_3 \). Hence \( W_1^* \cap W_2^* \cap W_3^* = \{ w_2^* \} \). Now, consider the second keyword of \( \text{inv} \geq (S_0) \) i.e. \( w_3 \). This keyword is in \( d_1 \) and \( d_2 \), but \( w_2 \) too. So \( W_1^* \cap W_2^* \cap W_3^* = \{ w_2^*, w_3 \} \), 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.

**Algorithm 5:** PowerSet Attack: keywords-tokens association.

**Input:** EDB, set \( S_0 \subseteq S \) of documents associated to their identifiers  
**Output:** Set of tokens \( W_{\text{rec}} \subseteq W^* \) associated to their keyword in \( W \)

\[
W_{\text{ign}}^* \leftarrow \emptyset; \\
// \text{Contains associated and indisting. tokens} \\
\text{Compute } \text{inv} \geq (S_0); \\
\text{foreach } w \in \text{inv} \geq (S_0) \text{ taken in decreasing order} \\
\quad \text{do} \\
\qquad A_w = \left( \bigcap \{ W_i^* \mid \text{id}_i \in \text{inv} \geq (S_0)[w] \} \right) \setminus W_{\text{ign}}^*; \\
\qquad W_{\text{ign}}^* = W_{\text{ign}}^* \cup A_w; \\
\quad // \text{Associated } (#A_w = 1) \text{ or indisting.} \\
\text{return } W_{\text{rec}} = \{ A_w \mid #A_w = 1 \}
\]

### 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 \( M_{L23} = \max_{d \in \text{DB}} | \{ \text{id} \mid \text{id} = \#d \} | \) and \( M_{L4} = \max_{d \in \text{DB}} | \{ \text{id} \mid \text{id} = \#d \} | \).

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

**Mask Attack.** For each known document \( d \) of \( S \), 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 \( O(\#S \cdot M_{L4}) \) 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 \( S \). Hence, constructing \( A_2 \) costs \( M_{L23}^2 \) applications of comask since we check all candidate pairs for the initial comask. We heuristically expect the sets \( A_1 \) to decrease as \( t \) grows. Indeed, if \( #A_2 \leq M_{L23} \), each association of \( d_1 \) starts from a smaller set \( A_{t-1} \) and imposes greater constraints, thus costing at most \( #A_2 \cdot M_{L23} \). In our experiments with chosen data sets, \( A_3 \) is almost always reduced to one element. Since, we check the comask for all candidates of each document \( d \) of \( S \) after the initialization, we conjecture a total complexity of \( O(M_{L23}^2 + #S \cdot M_{L23}) \) 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 \( S \). Hence, constructing \( A_2 \) costs \( M_{L23}^2 \) intersections cardinal 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 \( O(M_{L23}^2 + #A_2 \cdot #S \cdot M_{L23}) \) intersections cardinal computations.

### 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 online⁴, 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 take 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 Commons⁵ and Apache Lucene which is used too in (Cash et al., 2015; Islam et al., 2012). The last data set is the Project Gutenberg⁶. 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-

---

⁴cs.cmu.edu/∼enron/  
⁵mail-archives.apache.org/mod_mbox/commons-user/  
⁶gutenberg.org/wiki/Main_Page

<table>
<thead>
<tr>
<th>Data sets</th>
<th>Content</th>
<th>#DB</th>
<th>#W</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commons</td>
<td>mailing list</td>
<td>28,997</td>
<td>230,893</td>
<td>5.910,562</td>
</tr>
<tr>
<td>Enron</td>
<td>emails</td>
<td>490,369</td>
<td>643,818</td>
<td>47,301,160</td>
</tr>
<tr>
<td>Gutenberg</td>
<td>books</td>
<td>21,602</td>
<td>2,853,955</td>
<td>91,261,811</td>
</tr>
<tr>
<td>Lucene</td>
<td>mailing list</td>
<td>58,884</td>
<td>394,481</td>
<td>7,952,794</td>
</tr>
</tbody>
</table>

Table 1: Characteristics of used data sets.
standard 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 $S$ 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 $S$. 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 recover 21% of keywords in 489 seconds. The impact on the knowledge of the protected data is illustrated with graphs in Figure 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). Results are presented in Appendix 8.

**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 $S$ 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.

6 GAP BETWEEN L2- AND L1-SSE

We discuss the gap for an adversary to recover client queries over 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 $F(i_1, \ldots, i_q) = \sum_{1 \leq i \leq q} \frac{|\#(EDB(w^*_i) \cap EDB(w^*_j))}{n} - C_W(w_i, w_j)|^2$, 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 recovering queries with the help of queries and a
Keyword/token associations over the dataset

Document recovered >80%  Document recovered >90%  Documents fully recovered

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 $(w_i, \#DB(w_i))_{1 \leq i \leq n}$ from which he can compute the set $\{\#DB(w_1), \ldots, \#DB(w_n)\} = \{\#EDB(w^n_1), \ldots, \#EDB(w^n_n)\}$.

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 $S$ known by the adversary. Fig. 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 attacks, Fig. 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 \( \alpha \) different tuples of identifiers corresponding to the power set computed from the sample \( S \). 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 \( m \times n \), where \( M_{ij} = 1 \) if \( w_i \in d_j \) and \( M_{ij} = 0 \) 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 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 \( \alpha \) 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 = \((q - p)/p\).

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 \( S \) 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 \( S \) 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


APPENDIX

We present the results for the Mask Attack with datasets Commons in Fig. 4, Enron in Fig. 5 and Gutenberg in Fig. 6. Results for the Co-Mask Attack with datasets Commons are presented in Fig. 7 and Gutenberg in Fig. 8. To finish, we expose results for the PowerSet Attack with dataset Commons in Fig. 9.
Graphs. Two informations are presented in each graph. The first information is the rate of associations between keywords and search tokens. This rate is calculated compared to the number of keywords present in the sample $I$, then it is calculated compared to the total number of keywords ($\#W$) in the dataset.

The second information is the number of documents recovered partially or totally. We give the number of documents recovered at least 80%, 90% and 100%.

Tables. The first column of each table is the percentage of plaintext documents known by the server. The second column represent the rate of associations between keywords and search tokens. Other columns present the number of documents recovered at least 70%, 80%, 90% and 100%.

Discussion. We summarize the efficiency between our three attacks.

L4 Due to the amount of leaked information, our attack on L4-SSE schemes are the most efficient. On average, we exploit at 99% information given by the plaintext documents known by the adversary.

L3 In spite of the lost of information on the frequency of keywords in documents, our attack on L3-SSE schemes are also efficient as the attack on L4-SSE schemes. In fact, we still exploit at 99% information given by the plaintext documents known by the adversary.

L2 When we do not know the order of keyword in documents, we can now associate on average 20% of them to their corresponding token. However, these 20% of keywords are present in most documents, that is why we can recover a large part of the documents.

<table>
<thead>
<tr>
<th>%</th>
<th>Rate</th>
<th># 70%</th>
<th># 80%</th>
<th># 90%</th>
<th># 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.99</td>
<td>0.93</td>
<td>0.74</td>
<td>0.26</td>
<td>0.03</td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>0.96</td>
<td>0.83</td>
<td>0.40</td>
<td>0.06</td>
</tr>
<tr>
<td>3</td>
<td>0.99</td>
<td>0.97</td>
<td>0.87</td>
<td>0.49</td>
<td>0.09</td>
</tr>
<tr>
<td>4</td>
<td>0.99</td>
<td>0.98</td>
<td>0.90</td>
<td>0.54</td>
<td>0.11</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.98</td>
<td>0.92</td>
<td>0.59</td>
<td>0.13</td>
</tr>
<tr>
<td>10</td>
<td>0.99</td>
<td>0.99</td>
<td>0.95</td>
<td>0.72</td>
<td>0.23</td>
</tr>
<tr>
<td>20</td>
<td>0.99</td>
<td>0.99</td>
<td>0.98</td>
<td>0.84</td>
<td>0.38</td>
</tr>
<tr>
<td>30</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.89</td>
<td>0.51</td>
</tr>
<tr>
<td>40</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.92</td>
<td>0.61</td>
</tr>
<tr>
<td>50</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.95</td>
<td>0.69</td>
</tr>
</tbody>
</table>

Figure 4: Rate of recovered keywords and of 70%, 80%, 90% and 100% recovered documents on Commons dataset for the Mask Attack.
Figure 5: Rate of recovered keywords and of 70%, 80%, 90% and 100% recovered documents on Enron dataset for the Mask Attack.

Figure 6: Rate of recovered keywords and of 70%, 80%, 90% and 100% recovered documents on Gutenberg dataset for the Mask Attack.
Figure 7: Rate of recovered keywords and of 70%, 80%, 90% and 100% recovered documents on Commons dataset for the Co-Mask Attack.

Figure 8: Rate of recovered keywords and of 70%, 80%, 90% and 100% recovered documents on Gutenberg dataset for the Co-Mask Attack.
Figure 9: Rate of recovered keywords and of 70%, 80%, 90% and 100% recovered documents on Commons dataset for the PowerSet Attack.