“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

VAL: Volume and Access Pattern Leakage-abuse Attack with Leaked Documents

Steven Lambregts[1], Huanhuan Chen[1], Jianting Ning[2,3](✉), and Kaitai Liang[1]

[1] Delft University of Technology, 2628 CD Delft, The Netherlands
{s.f.lambregts, h.chen-2, kaitai.liang}@tudelft.nl
[2] Singapore Management University, Singapore 188065
[3] Fujian Normal University, Fuzhou 350117, China
jtning88@gmail.com

Abstract. Searchable Encryption schemes provide secure search over encrypted databases while allowing admitted information leakages. Generally, the leakages can be categorized into `access` and `volume pattern`. In most existing SE schemes, these leakages are caused by practical designs but are considered an acceptable price to achieve high search efficiency. Recent attacks have shown that such leakages could be easily exploited to retrieve the underlying keywords for search queries. Under the umbrella of attacking SE, we design a new Volume and Access Pattern Leakage-Abuse Attack (VAL-Attack) that improves the matching technique of LEAP (CCS ’21) and exploits both the `access and volume patterns`. Our proposed attack only leverages leaked documents and the keywords present in those documents as auxiliary knowledge and can effectively retrieve document and keyword matches from leaked data. Furthermore, the recovery performs without false positives. We further compare VAL-Attack with two recent well-defined attacks on several real-world datasets to highlight the effectiveness of our attack and present the performance under popular countermeasures.

Keywords: Searchable Encryption · Access pattern · Volume pattern · Leakage · Attack

1 Introduction

In practice, to protect data security and user privacy (e.g., under GDPR), data owners may choose to encrypt their data before outsourcing to a third-party cloud service provider. Encrypting the data enhances privacy and gives the owners the feeling that their data is stored safely. However, this encryption relatively restricts the searching ability. Song et al. [34] proposed a Searchable Encryption (SE) scheme to preserve the search functionality over outsourced and encrypted data. In the scheme, the keywords of files are encrypted, and when a client wants to query a keyword, it encrypts the keyword as a token and sends it to the server. The server then searches the files with the token corresponding to the query, and afterwards, it returns the matching files. Since the seminal SE scheme, many research works have been presented in the literature, with symmetrical [7, 9, 10, 13] and asymmetrical encryption [1, 5, 36, 38]. Nowadays, SE schemes have been deployed in many real-world applications such as ShadowCrypt [17] and Mimesis Aegis [23].

Leakage. In an SE scheme, an operational interaction is usually defined as a client sending a query to the server and the server responding to the query with the matching files. Nevertheless, this interaction could be eavesdropped on by an attacker. The messages could be intercepted because they are sent over an unprotected channel, or the attacker is the cloud service provider itself, who stores and accesses all the search requests and responses. The attacker may choose to match the query with a keyword such that he can comprehend what information is present on

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

the server. The query and response here are what we may call *leakage*. In this work, we consider two main types of *leakage patterns*: the `access pattern`, the response from the server to a query, and the `search pattern`, which is the frequency a query is sent to the server. Besides these types, we also consider the `volume pattern` as leakage. This pattern is seen as the size of the stored documents on the server. The leakage patterns can be divided into four levels, by Cash et al. [8]. In this work, we consider our leakage level to be L2, which equals the fully-revealed occurrence pattern, together with the volume pattern to create a new attack on the SE scheme. Note that a formal definition of the leakages is given in Section 3.1.

Attacks on SE. There exist various attacks on SE that work and perform differently. Most of these attacks take the leaked files as auxiliary knowledge. Islam et al. [18] presented the foundation for several attacks on SE schemes. They stated that, with sufficient auxiliary knowledge, one could create a co-occurrence matrix for both the leakage and the knowledge so that it can easily map queries to the keywords based on the lowest distance. Cash et al. [8] later proposed an attack where the query can be matched to a particular keyword based on the total occurrence in the leaked files. These attacks with knowledge about some documents are known as *passive attacks with pre-knowledge*. Blackstone et al. [4] developed a Subgraph$_{\mathrm{VL}}$ attack that provides a relatively high query recovery rate even with a small subset of the leaked documents. The attack matches keywords based on unique document volumes as if it is the response pattern. Ning et al. [28] later designed the LEAP attack. LEAP combines the existing techniques, such as co-occurrence and the unique number of occurrences, to match the leaked files to server files and the known keywords to queries based on unique occurrences in the matched files. It makes good use of the unique count from the Count attack [8], a co-occurrence matrix from the IKK attack [18] (although LEAP inverts it to a document co-occurrence matrix) and finally, unique patterns to match keywords and files. Note that we give related work and general comparison in Section 6.

Limitations. The works in [4, 8, 18, 28] explain their leakage-abusing methods, but they only abuse a single leakage pattern, while multiple are leaked in SE schemes. Besides the leakage patterns, the state-of-the-art LEAP attack abuses the `access pattern` but does not exploit its matching techniques to the full extent. In addition to extending their attack, a combination of leakage can be used to match more documents and queries.

We aim to address the issue of matching keywords by exploiting both the `access pattern` and `volume pattern`. The following question arises naturally:

Could we match queries and documents in a passive attack by exploiting the volume and access patterns to capture a high recovery rate against popular defences?

Contributions. We answer the above research question by designing an attack that matches leaked files and keywords. Our attack expands the matching techniques from the LEAP attack [28] and exploits the `volume pattern` to match more documents. The attack improves the LEAP attack by fully exploring the leakage information and combining the uniqueness of document volume to match more files. These matches can then be used to extract keyword matches. All the matches found are correct, as we argue that false positives are not valuable in real-world attacks.

- Besides exploiting the `access pattern`, we also abuse `volume pattern` leakage. We match documents based on a unique combination of volume and number of keywords with both leakage patterns. We can match almost all leaked documents to server documents using this approach.
- We match keywords using their occurrence pattern in matched files.
- Besides matching keywords in matched files, we use all leaked documents for unique keyword occurrence, expanding the keyword matching technique from the LEAP attack. We do this to get the maximum amount of keyword matches from the unique occurrence pattern.

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

• We run our attack against three different datasets to test the performance, where we see that the results are outstanding as we match almost all leaked documents and a considerable amount of leaked keywords. Finally, we compare our attack to the existing state-of-the-art LEAP and Subgraph$_{\mathrm{VL}}$ attacks. Our attack performs great in revealing files and underlying keywords. In particular, it surpasses the LEAP attack, revealing significantly more leaked files and keywords. VAL-Attack recovers almost 98% of the known files and above 93% of the keyword matches available to the attacker once the leakage percentage reaches 5%. When 10% of the Enron database is leaked, which is 3,010 files with 4,962 keywords, we match 2,950 files and 4,909 queries, respectively, corresponding to 98% and 99%. VAL-Attack can still compromise encrypted information, e.g., over 90% recovery (with 10% leakage) under volume hiding in Enron and Lucene, even under several popular countermeasures. We note that our proposed attack is vulnerable to a combination of padding and volume hiding.

2 Preliminaries

2.1 Searchable Encryption

In a general SE scheme, a user encrypts her data and uploads the encrypted data to a server. After uploading the data, the user can send a query containing an encrypted keyword to the server, and the server will then respond with the corresponding data. We assume the server is honest-but-curious, meaning that it will follow the protocol but will try to retrieve as much information as possible.

The scheme. At a high level, an SE scheme consists of three polynomial-time algorithms: Enc, QueryGen and Search [13, 15, 21, 24, 27]. Definition 1 shows the scheme in more detail. The client runs the algorithm Enc and encrypts the plaintext documents and the corresponding keywords before uploading them to the server. Enc outputs an encrypted database EDB, which is sent to the server. QueryGen, run by the user, requires a keyword and outputs a query token that can be sent to the server. The function Search is a deterministic algorithm that is executed by the server. A query q is sent to the server; the server takes the encrypted database EDB and returns the corresponding identifiers of the files $EDB(q)$. After it has retrieved the file identifiers, the user has to do another interaction with the server to retrieve the actual files.

Definition 1. *A searchable encryption scheme includes three algorithms {Enc, QueryGen, Search} that operate as follows:*

- *$Enc(K, F)$: the encryption algorithm takes a master key K and a document set $F = \{F_1, ..., F_n\}$ as input and outputs the encrypted database $EDB := \{Enc_k(F_1), ..., Enc_K(F_n)\}$;*
- *$QueryGen(w)$: the query generation algorithm takes a keyword w as input and outputs a query token q.*
- *$Search(q, EDB)$: the search algorithm takes a query q and the encrypted database EDB as input and outputs a subset of the encrypted database EDB, whose plaintext contains the keyword corresponding to the query q.*

Leakage. A query and the server response are considered the `access pattern`. The documents passed over the channel have their volume; this information is considered the `volume pattern`. In Section 3.1, we will explain the leakage in more detail.

2.2 Notation

In the VAL-Attack, we have m' keywords (w) and m queries (q), and n' leaked-documents and n server documents, denoted as d_i and ed_i, respectively; for a single document, similarly for w_i and q_i. Note w_i may not be the underlying keyword for query q_i, equal for d_i and ed_i. The notations are given in Table 1.

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

Table 1: Notation Summary

F	Plaintext document set, $F = \{d_1, ..., d_n\}$	F'	Leaked document set, $F' = \{d_1, ..., d_{n'}\}$
E	Server document set, $E = \{ed_1, ..., ed_n\}$	W	Keyword universe, $W = \{w_1, ..., w_m\}$
W'	Leaked keyword set, $W' = \{w_1, ..., w_{m'}\}$	Q	Query set, $Q = \{q_1, ..., q_m\}$
A	$m' \times n'$ matrix of leaked documents	B	$m \times n$ matrix of server documents
M'	$n' \times n'$ co-occurrence matrix of F'	M	$n \times n$ co-occurrence matrix of E
v_i	Volume (bit size) of document i	$\|d_i\|$	Number of keywords in document i
C	Set of matched documents	R	Set of matched queries

3 Models

In an ideal situation, there is no information leaked from the encrypted database, the queries sent, or the database setup. Unfortunately, such a scheme is not practical in real life as it costs substantial performance overheads [16]. The attacker and the leakage are two concerns in SE schemes, and we will discuss them both in the following sections, as they can vary in different aspects.

3.1 Leakage Model

Leakage is what we define as information that is (unintentionally) shared with the outer world. In our model, the attacker can intercept everything sent from and to the server. The attacker can intercept a query that a user sends to the server and the response from the server. It then knows which document identifiers correspond to which query. This *query → document identifier* response is what we call the `access pattern`. The leakage is defined as [4]:

Definition 2 (`access pattern`). *The function access pattern (AP)* $= (AP_{k,t})_{k,t \in \mathbb{N}} : F(k) \times W^t(k) \rightarrow [2^{[n]}]^t$*, such that* $AP_{k,t}(D, w_1, ..., w_t) = D(w_1), ..., D(w_t)$.

As discussed earlier, we assume the leakage level is L2 [8], where the attacker does not know the frequency or the position of the queried keywords in the document response.

The `volume pattern` is leakage that tells the size of the document. It is relevant to all response leaking encryption schemes [6, 9, 11, 13, 20, 21] and ORAM-based SE schemes [26]. The leakage is defined formally as follows [4]:

Definition 3 (`volume pattern`). *The function volume pattern (Vol)* $= (Vol_{k,t})_{k,t \in \mathbb{N}} : F(k) \times W^t(k) \rightarrow \mathbb{N}^t$*, such that* $Vol_{k,t}(D, w_1, ..., w_n) = ((|d|_w)_{d \in D(w_1)}, ..., (|d|_w)_{d \in D(w_n)})$*, where* $|\cdot|_w$ *represents the volume in bytes.*

3.2 Attack Model

The attacker in SE schemes can be a malicious server that stores encrypted data. Since the server is honest-but-curious [4], it will follow the encryption protocol but wants to learn as much as possible. Therefore, the attacker is passive but still eager to learn about the content present on the server. Our attacker has access to some leaked plaintext documents, keeps track of the `access and volume pattern` and tries to reveal the underlying server data. Fig. 1 shows a visualization of our attack model. We assume that the attacker has access to all the queries

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

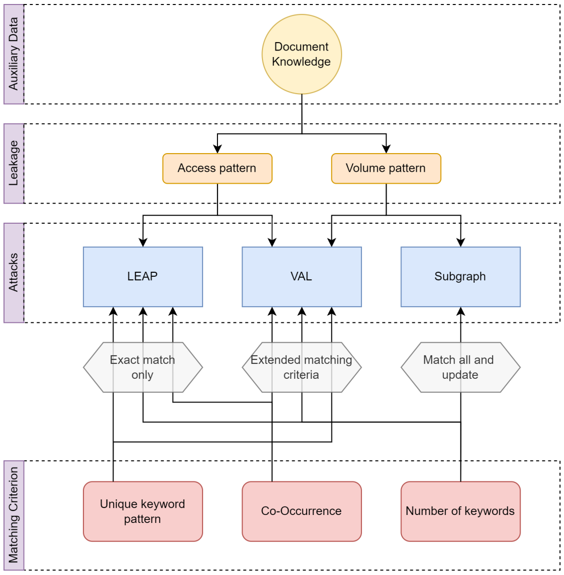


Figure 1: Technical Framework of Existing Attacks

and responses used in the SE scheme. This number of queries is realistic because if one waits long enough, all the queries and results will eventually be sent over the user-server channel. The technical framework delineates the LEAP, $\text{Subgraph}_{\text{VL}}$ and our designed attack.

The attacker in our model has access to some unencrypted files stored on the server. This access can be feasible because of a security breach at the setup phase of the scheme, where the adversary can access the revealed files. Another scenario is if a user wants to transfer all of his e-mails from his unencrypted mail storage to an SE storage server. The server can now access all the original mail files, but new documents will come as new e-mails arrive. Therefore, the adversary has partial knowledge about the encrypted data present on the server. The attacker has no access to any existing query to keyword matches and only knows the keywords present in the leaked files. With this information, the attacker wants to match as many encrypted document identifiers to leaked documents and queries to keywords such that he can understand what content is stored on the server.

The passive attacker is less potent than an active attacker, who can upload documents, with chosen keywords, to the server to match queries to keywords [37]. Furthermore, the attacker has no access to the encryption or decryption oracle. Because the attacker relies on the `access and volume pattern` countermeasures that hide these patterns will reduce the attack performance.

4 The Proposed Attack

4.1 Main Idea

At a high level, our attack is built from the LEAP attack [28] by elevating the keyword matching metric to increase the number of keyword matches. Furthermore, each document is labelled with its document volume and number of keywords, and VAL-attack matches using the uniqueness of this label, improving the recovery rate. We first extend the matching technique from LEAP. The approach does not consist of only checking within the matched documents but also keeping

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

track of the occurrence in the unmatched files. This method results in more recovered keywords for the improvement of LEAP that provides a way to match rows that do not uniquely occur in the matched files. We expand the attack by exploiting the `volume pattern` since the document size is also leaked from response leaking encryption schemes, as described in Section 3.1. We can extend the comprehensive attack by matching documents based on the `volume pattern`. Our new attack fully explores the leakage information and matches almost all leaked documents. We increase the keyword matches with the maximal file matches to provide excellent performance.

4.2 Leaked Knowledge

The server stores all the documents in the scheme. There are a total of n plaintext files denoted as the set $F = \{d_1, ..., d_n\}$, with in total m keywords, denoted as the set $W = \{w_1, ..., w_m\}$. We assume the attacker can access:

- The total number of leaked files (i.e. plaintext files) is n' with in total m' keywords. Suppose $F' = \{d_1, ..., d_{n'}\}$ is the set of documents known to the attacker and $W' = \{w_1, ..., w_{m'}\}$ is the corresponding set of keywords that are contained in F'. Note that $n' \leq n$ and $m' \leq m$.
- The set of encrypted files, denoted as, $E = \{ed_1, ..., ed_n\}$ and corresponding query tokens, $Q = \{q_1, ..., q_m\}$ with underlying keyword set W.
- The volume of each server observed document or leaked file is denoted as v_x for document d_x or server document ed_x. The number of keywords or tokens is represented as the size of the document $|d_x|$ or $|ed_x|$ for the same documents, respectively.

The attacker can construct an $m' \times n'$ binary matrix A, representing the leaked documents and their corresponding keywords. $A[d_x][w_y] = 1$ iff. keyword w_y occurs in document d_x. The dot product of A is denoted as the symmetric $n' \times n'$ matrix M', whose entry is the number of keywords that are contained in both document d_x and document d_y. We give an example of the matrices with known documents in Fig. 6 (Appendix A).

After observing the server's files and query tokens, the attacker can construct an $m \times n$ binary matrix B, representing the encrypted files and related query tokens. $B[ed_x][q_y] = 1$ iff. query q_y retrieved document ed_x. The dot product of B is denoted as the symmetric $n \times n$ matrix M, whose entry is the number of query tokens that retrieve files ed_x and ed_y from the server. We give an example of the matrices with observed encrypted documents in Fig. 7 (Appendix A).

4.3 Our Design

The basis of the attack is to recursively find row and column mappings between the two created matrices, A and B, where a row mapping represents the underlying keyword of a query sent to the server, and a column mapping indicates the match between a server document identifier and a leaked plaintext file. Note that each leaked document is still present on the server, meaning that $n' \leq n$ and there is a matching column in B for each column in A. Similarly to the rows, each known keyword corresponds to a query, so $m' \leq m$ as we could know all the keywords, but we do not know for sure. In theory, there is a correct row mapping for each row in A to a row in B. The goal of the VAL-Attack is to find as many correct mappings as possible.

We divide the process of finding as many matches as possible into several steps. The first step is to prepare the matrices for the rest of the process. The algorithm then maps columns based on unique column-sum, as they used in the Count attack [8], but instead of using it on keywords, we try to match documents here. Another step is matching documents based on unique volume and the number of keywords or tokens. As this combination can be a unique pattern, we can match many documents in this step. The matrices M and M' are used to match documents based on co-occurrence. Eventually, we can pair keywords on unique occurrences in the matched

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

documents when several documents are matched. This technique is used in the Count attack [8], but we ’simulate’ our own 100% knowledge here. With the matched keywords, we can find more documents, as these will give unique rows in matrices A and B that can be matched. We will introduce these functions in detail in the following paragraphs.

Initialization. First, we initialize the algorithm by creating two empty dictionaries, to which we eventually add the correct matches. We create one dictionary for documents and the other for the matched keywords, C (for column) and R (for row). Next, as we want to find unique rows in the matrices A and B, we must extend matrix A. It could be possible that not all underlying keywords are known beforehand, in which case $n' < n$, and we have to extend matrix A to find equal columns. Therefore we extend matrix A to an $m \times n'$ matrix that has the first m' rows equal to the original matrix A and the following $m - m'$ rows of all 0s. See Fig. 10 (Appendix A) for an example. The set $\{w_{m'+1}, ..., w_m\}$ represents the keywords that do not appear in the leaked document set F'.

Number of keywords. Now that the number of rows in A and B are equal, we can find unique column-sums to match documents. This unique sum indicates that a document has a unique number of keywords and can thus be matched based on this unique factor. Similar to the technique in the Count attack [8], we sum the columns, here representing the keywords in A and B. The unique columns in B can be matched to columns in A, as they have to be unique in A as well. If a column$_j$-sum of B is unique and column$_{j'}$-sum of A exists, we can match documents ed_j and $d_{j'}$ because they have the same unique number of keywords.

Volume and keyword pattern. The next step is matching documents based on volume and keyword pattern. If there is a server document ed_j with a unique combination of volume v_j and number of tokens $|ed_j|$ and there is a document $d_{j'}$ with the same combination, we can match document ed_j to $d_{j'}$. However, if multiple server documents have the same pattern, we need to check for unique columns with the already matched keywords between these files. Initially, we will have no matched keywords, but we will rerun this step later in the process. Fig. 2 shows a concrete example, and Algorithm 1 describes our method.

Figure 2: Document matching on volume and number of keywords. Given multiple candidates, match on a unique column with the already matched keywords.

(a) Multiple documents with the same pattern of volume and number of keywords/tokens.

Leaked files	$\cdots$	d_4	d_6	d_8	$\cdots$	$d_{n'}$
Volume	$\cdots$	120	120	120	$\cdots$	120
#Keywords	$\cdots$	15	15	15	$\cdots$	18

Server files	$\cdots$	ed_6	ed_9	ed_{10}	$\cdots$	ed_n
Volume	$\cdots$	120	120	120	$\cdots$	150
#Tokens	$\cdots$	20	15	15	$\cdots$	15

(b) With the already matched keywords, create unique columns to match documents. Here d_6 and ed_8 can be matched, as well as d_9 and ed_{15}.

$$
\begin{array}{c c}
A_{CR} & \begin{array}{cccc} d_4 & d_6 & d_8 & d_9 \end{array} \\
\begin{array}{c} w_2 \\ w_3 \\ w_5 \\ \vdots \\ w_t \end{array} & \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \\ 1 & 1 & 1 & 0 \end{pmatrix}
\end{array}
\qquad
\begin{array}{c c}
B_{CR} & \begin{array}{cccc} ed_8 & ed_9 & ed_{10} & ed_{15} \end{array} \\
\begin{array}{c} q_1 \\ q_3 \\ q_{15} \\ \vdots \\ q_t \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \\ 1 & 1 & 1 & 0 \end{pmatrix}
\end{array}
$$

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

Algorithm 1 matchByVolume

```
**Input**: $R, A\ (m \times n'), B\ (m \times n)$
$C' \leftarrow \{\}$
patterns $\leftarrow \{(v_j, |ed_j|)$ with volume $v_j$ and #tokens $|ed_j|$ of document $ed_j\}$
for $p \in$ patterns do
    enc_docs $\leftarrow [ed_j$ with pattern $p]$
    if $|$enc_docs$| = 1$ then
        $ed_j \leftarrow$ enc_docs$[0]$
        $C'[ed_j] \leftarrow d_{j'}$ with pattern $p$
    else if $|R| > 0$ then
        docs $\leftarrow [d_{j'}$ with pattern $p]$
        $B_{CR} \leftarrow$ enc_docs columns and $R$ rows of $B$
        $A_{CR} \leftarrow$ docs columns and $R$ rows of $A$
        for column$_j \in B_{CR}$ that is unique do
            $C'[ed_j] \leftarrow d_{j'}$ with column$_j \in A_{CR}$
return $C'$
```

Co-occurrence. When having some matched documents, we can use the co-occurrence matrices M and M' to find other document matches. For an unmatched server document ed_x, we can try an unmatched leaked document d_y. If $M_{x,k}$ and $M'_{y,k'}$ are equal for each matched document pair $(ed_k, d_{k'})$ and no other document $d_{y'}$ has the same results, then we have a new document match between ed_x and d_y. The algorithm for this step is shown in Algorithm 2.

Algorithm 2 coOccurrence

```
**Input**: $C, M\ (n \times n), M\ (n' \times n), A\ (m \times n'), B\ (m \times n)$
while $C$ is increasing do
    for each $d_{j'} \notin C$ do
        sum$_{j'} \leftarrow$ column$_{j'}$-sum of $A$
        candidates $\leftarrow [ed_j \notin C$ where column$_j$-sum of $B =$ sum$_{j'}]$
        for $ed_j \in$ candidates do
            for $(ed_k, d_{k'}) \in C$ do
                if $M_{j,k} \neq M'_{j',k'}$ then
                    candidates $\leftarrow$ candidates $\setminus ed_j$
        if $|$candidates$| = 1$ then
            $ed_j \leftarrow$ candidates$[0]$
            $C[ed_j] \leftarrow d_{j'}$
return $C$
```

Keyword matching. We match keywords using the matched documents. To this end, we create matrices B_c and A_c by taking the columns of matched documents from matrices B and A. Note that these columns will be rearranged to the order of the matched documents, such that column B_{c_j} is equal to column $A_{c_{j'}}$ for document match $(ed_j, d_{j'})$. Matrices B_c and A_c are shaped $m \times t$ and $m' \times t$, respectively, for t matched documents. We give the algorithm for this segment in Algorithm 3 and a simple example in Fig. 8 (Appendix A).

A row in the matrices indicates in which documents a query or keyword appears. If a row$_i$ in B_c is unique, row$_i$ is also unique in B, similar to A_c and A. Hence, for row$_i$ in B_c, that is unique, and if there is an equal row$_j$ in A_c, we can conclude that the underlying keyword of q_i is w_j.

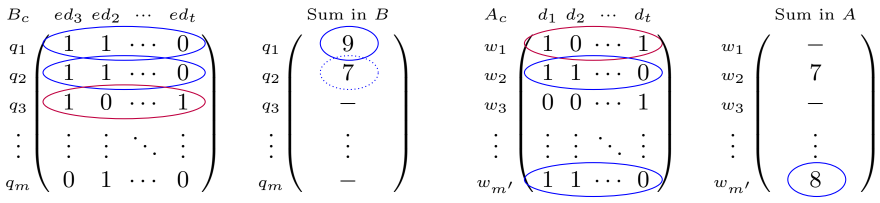
“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

Algorithm 3 matchKeywords

```
    Input: $C, A\ (m \times n'), B\ (m \times n)$
    $R \leftarrow \{\}$
    $B_c \leftarrow C$ columns of $B$
    $A_c \leftarrow C$ columns of $A$
    for $\text{row}_i \in B_c$ do
        if $\text{row}_i$ is unique in $B_c$ then
            if $\text{row}_{i'} \in A_c = \text{row}_i$ then
                $R[q_i] \leftarrow w_{i'}$
        else                              ▷ Match based on occurrence in (server) files
            docs $\leftarrow [i' \in A_c$ where $A_c[i'] = \text{row}_i]$
            e_docs $\leftarrow [j \in B_c$ where $B_c[j] = \text{row}_i]$
            $B_x \leftarrow$ sum of rows in $B$[e_docs], sort descending
            $A_x \leftarrow$ sum of rows in $A$[docs], sort descending
            if $B_x[1] < A_x[0] < B_x[0]$ then
                $i_x \leftarrow$ index of $B_x[0] \in$ e_docs
                $j_x \leftarrow$ index of $A_x[0] \in$ docs
                $R[q_{i_x}] \leftarrow w_{j_x}$
    return $R$
```

Nevertheless, if row_i is not unique in B_c, we can still try to match the keyword to a query. A keyword can occur more often in the unmatched documents than their query candidates; thus, they will not be valid candidates. We create a list B_x with for each similar row_i in B_c the sum of row_i in B; similar for list A_x, with row_i in A_c and the sum of row_i in A. Next, if the highest value of A_x, which is A_{x_j}, is higher than the second-highest value of B_x, we can conclude that keyword w_j corresponds to the highest value of B_x, i.e. B_{x_j}, which means that w_j matches with q_j. We put an example in Fig. 3.

Figure 3: Example of matching keywords in matched documents. Query q_3 has a unique row and therefore matches with keyword w_1. Queries q_1, q_2 and keywords w_2, $w_{m'}$ have the same row. However, keyword $w_{m'}$ occurs more often in A than w_2 and query q_2 in B. Therefore q_1 matches with $w_{m'}$.



$$\begin{array}{c|cccc} B_c & ed_3 & ed_2 & \cdots & ed_t \\ \hline q_1 & 1 & 1 & \cdots & 0 \\ q_2 & 1 & 1 & \cdots & 0 \\ q_3 & 1 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_m & 0 & 1 & \cdots & 0 \end{array} \qquad \begin{array}{c|c} & \text{Sum in } B \\ \hline q_1 & 9 \\ q_2 & 7 \\ q_3 & - \\ \vdots & \vdots \\ q_m & - \end{array} \qquad \begin{array}{c|cccc} A_c & d_1 & d_2 & \cdots & d_t \\ \hline w_1 & 1 & 0 & \cdots & 1 \\ w_2 & 1 & 1 & \cdots & 0 \\ w_3 & 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m'} & 1 & 1 & \cdots & 0 \end{array} \qquad \begin{array}{c|c} & \text{Sum in } A \\ \hline w_1 & - \\ w_2 & 7 \\ w_3 & - \\ \vdots & \vdots \\ w_{m'} & 8 \end{array}$$

Keyword order in documents. We aim to find more documents based on unique columns given the query and keyword mappings. First, we create matrices B_r and A_r with the rows from the matched keywords in R. B_r and A_r are submatrices of B and A, respectively, with rearranged row order. B_r and A_r are shaped $t \times n$ and $t \times n'$, respectively, for t matched keywords. Note that we show an example in Fig. 9 (Appendix A). If any column_j of B_r is unique and there exists an equal $\text{column}_{j'}$ in A_r, we know that ed_j is a match with $d_{j'}$.

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

The next step is to set the rows of the matched keywords to 0 in B and A. Then, similar to before, we use the technique from the Count attack [8]; we sum the updated columns in A and B and try to match the unique columns in B to columns in A. If a column$_j$-sum of B is unique and an equal column$_{j'}$-sum in A exists, we can match document ed_j and $d_{j'}$.

The complete algorithm of our VAL-attack is in Algorithm 4, Appendix B.

4.4 Countermeasure Discussions

Some countermeasures have been proposed to mitigate leakage-abuse attacks [8, 12, 18, 32]. The main approaches are padding and obfuscation. Below, we have some discussions on the countermeasures.

The IKK attack [18] and the Count attack [8] discussed a padding countermeasure, where they proposed a technique to add fake document identifiers to a query response. These false positives could then later be removed by the user. This technique is also called *Hiding the Access Pattern* [22].

The LEAP attack [28] crucially relies on the number of keywords per document, and if the scheme adds fake query tokens to documents on the server, they will not be able to match with their known documents. However, they also proposed a technique that describes a modified attack that is better resistant to padding. This technique, which is also used in the Count attack [8], makes use of a window to match keywords. But this will give false positives and thus reduce the performance of the attack.

The Subgraph$_{\mathrm{VL}}$ attack [4] depends on the volume of each document. Volume-hiding techniques from Kamara et al. [19] reduce the attack's performance, but it is not clear if they completely mitigate the attack.

A padding technique that will make all documents of the same size, i.e. adding padding characters, will reduce the uniqueness in matching based on the volume of a document. If the padding technique can be extended such that false positives are added to the `access pattern`, we have no unique factor in matching documents based on the number of keywords per file. Therefore, a combination of the two may decrease the performance of the VAL-Attack.

5 Evaluation

We set up the experiments to run the proposed attack to evaluate the performance. Furthermore, we compare the file and query recovery of the VAL-Attack with the results from the LEAP [28] and Subgraph$_{\mathrm{VL}}$ attack [4]. We notice that the LEAP attack is not resistant to the test countermeasures, and Blackstone et al. [4] argue for their Subgraph$_{\mathrm{VL}}$ attack that it is not clear whether volume-hiding constructions may mitigate the attack altogether. From this perspective, we only discuss the performance of VAL-Attack against countermeasures in Section 5.3. It would be an interesting problem to test the countermeasures on the LEAP and Subgraph$_{\mathrm{VL}}$ attacks, but that is orthogonal to the focus of this work.

5.1 Experimental Setup

We used the Enron dataset [35] to run our comparison experiments. We leveraged the *_sent_-mail* folder from each of the 150 users from this dataset, resulting in 30,109 e-mails from the Enron corporation. The second dataset we used is the Lucene mailing list [2]; we specifically chose the ”java-user” mailing list from the Lucene project for 2002-2011. This dataset contains 50,667 documents. Finally, we did the tests on a collection of Wikipedia articles. We extracted

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

plaintext documents from Wikipedia in April 2022 using a simple wiki dump[4] and used the tool from David Shapiro [33] to extract plaintext data, resulting in 204,737 files. The proposed attack requires matrices of size $n \times n$; therefore, we limited the number of Wikipedia files to 50,000. We used Python 3.9 to implement the experiments and run them on machines with different computing powers to improve running speed.

To properly leverage those data from the datasets for the experiments, we first extracted the information of the Enron and Lucene e-mail content. The title's keywords, the names of the recipients or other information present in the e-mail header were not used for queries. NLTK corpus [3] in Python is used to get a list of English vocabulary and stopwords. We removed the stopwords with that tool and stemmed the remaining words using Porter Stemmer [30]. We further selected the most frequent keywords to build the keyword set for each document. For each dataset, we extracted 5,000 words as the keyword set W. Within the Lucene e-mails, we removed the unsubscribe signature because it appears in every e-mail.

The server files (n) and keywords (m) are all files from the dataset and 5,000 keywords, respectively. The leakage percentage determines the number of files (m') known to the user. The attacker only knows the keywords (n') leaked with these known documents. The server files and queries construct a matrix B of size $m \times n$; while the matrix A of size $m' \times n'$ is constructed with the leaked files. We took the dot product for both matrices and created the matrices M and M', respectively. Note that the source code to simulate the attack and obtain our results is available here: https://github.com/StevenL98/VAL-Attack.

Because our attack does not create false positives, the accuracy of the retrieved files and keywords is always 100%. Therefore, we calculated the percentage of files and keywords retrieved from the total leaked files and keywords. Each experiment is run 20 times to calculate an average over the simulations. We chosen 0.1%, 0.5%, 1%, 5%, 10%, 30% as leakage percentages. The lower percentages are chosen to compare with the results from the LEAP attack [28], and the maximum of 30% is chosen because of the stagnation in query recovery.

5.2 Experimental Results

The results tested with the different datasets are given in Fig. 4a and Fig. 4b, which show the number and percentage of files and keywords recovered by our attack. The solid line is the average recovery in those plots, and the shades are the error rate over the 20 runs.

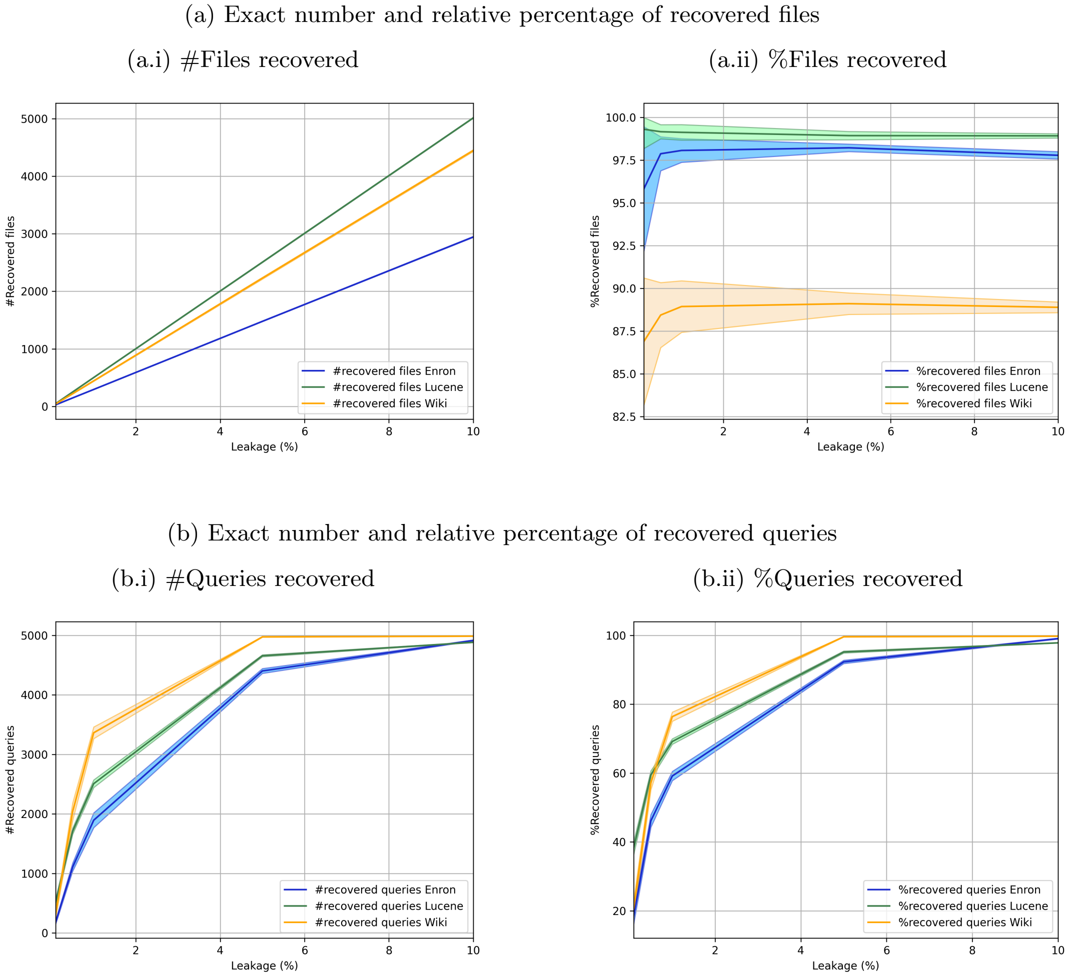
We can see that the VAL-attack recovers almost 98% of the known files and above 93% of the keywords available to the attacker once the leakage percentage reaches 5%. These percentages are based on the leaked documents. When 10% of the Enron database is leaked, which is 3,010 files with 4,962 keywords, we can match 2,950 files and 4,909 queries, corresponding to 98% and 99%, respectively. The Lucene dataset is more extensive than Enron, and therefore we have more files available for each leakage percentage. One may see that we can recover around 99% of the leaked files and a rising number of queries, starting from 40% of the available keyword set. The Wikipedia dataset does not consist of e-mails but rather lengthy article texts. We reveal fewer files than the e-mail datasets, but we recover just below 90% of the leaked files, and from 1% leakage, we recover more available keywords than the other datasets. This difference is probably because of the number of keywords per file since the most frequent keywords are chosen.

With the technique we proposed, one can match leaked documents to server documents for almost all leaked documents. Next, the algorithm will compute the underlying keywords to the queries. It is up to the attacker to allow false positives and improve the number of (possible) correctly matched keywords, but we decided not to include it.

[4] https://dumps.wikimedia.org/simplewiki/20220401/simplewiki-20220401-pages-meta-current.xml.bz2

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

Figure 4: Results for VAL-Attack, with the actual number and the percentage of recovered files and queries for different leakage percentages.



(a) Exact number and relative percentage of recovered files

(a.i) #Files recovered

(a.ii) %Files recovered

(b) Exact number and relative percentage of recovered queries

(b.i) #Queries recovered

(b.ii) %Queries recovered

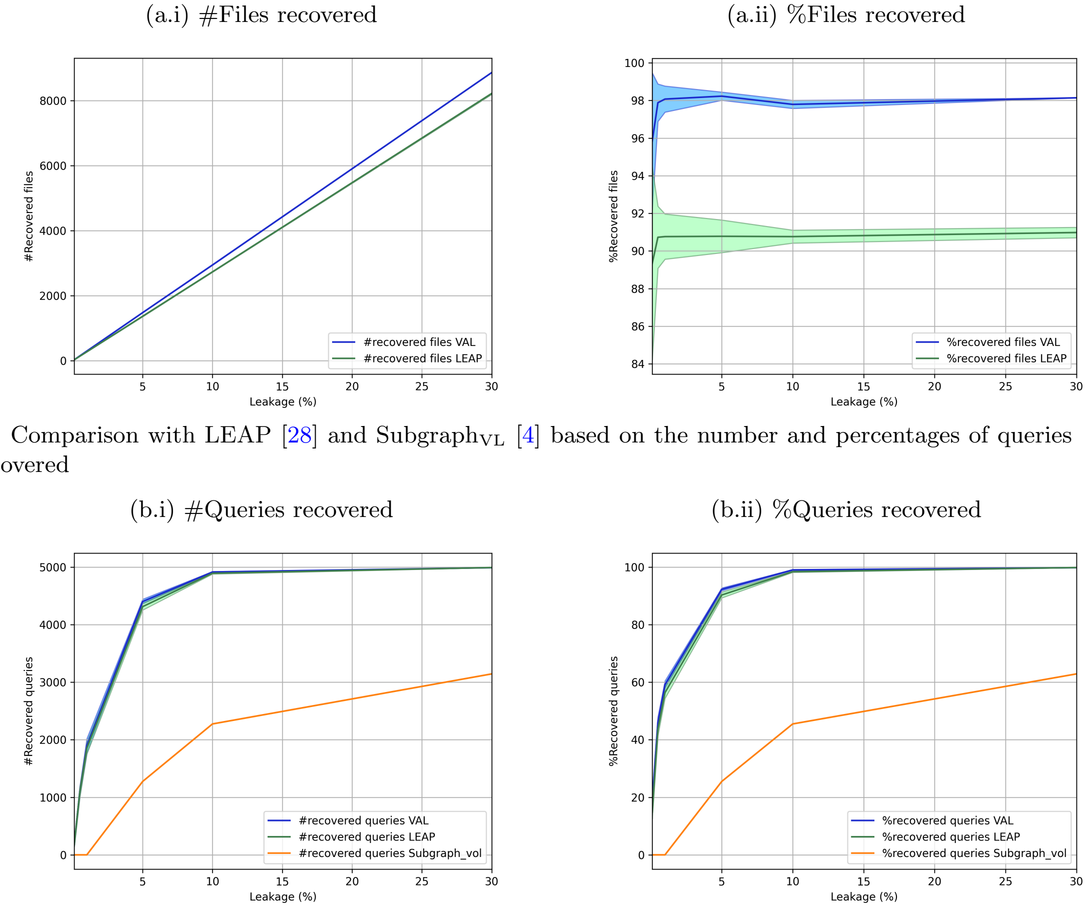
Comparison. We compare the performance of VAL-Attack to two attacks with the Enron dataset. One is the LEAP attack [28] (which is our cornerstone), while the other is the Subgraph$_{\text{VL}}$ attack [4] (as they use the volume pattern as leakage). We divide the comparison into two parts: the first is for recovering files, and the second is for queries recovery.

As shown in Fig. 5, we recover more files than the LEAP attack, and the gap in files recovered expands as the leakage percentage increases, see Fig. 5a.i. The difference in the percentage of files recovered is stable, as VAL-Attack recovers about eight percentage points more files than the LEAP attack, see Fig. 5a.ii. The comparison outcome for recovered queries can be seen in Fig. 5b. We can see that the recovered queries do not show a significant difference with the LEAP attack as that attack performs outstandingly in query recovery. The most significant difference is around 5% leakage, where VAL-Attack retrieves around 100 queries more than the LEAP attack, which could influence a real-world application. Compared to the Subgraph$_{\text{VL}}$, we see in Fig. 5b.ii that the combination of the `access pattern` and the `volume pattern` is a considerable improvement; we reveal about 60 percentage points more of the available queries.

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

Figure 5: Comparison of VAL-Attack

(a) Comparison with LEAP [28] based on the number and percentages of files recovered



(a.i) #Files recovered

(a.ii) %Files recovered

(b) Comparison with LEAP [28] and Subgraph$_{\mathrm{VL}}$ [4] based on the number and percentages of queries recovered

(b.i) #Queries recovered

(b.ii) %Queries recovered

5.3 Countermeasure Performance

As discussed in Section 4.4, there are several options for countermeasures against attacks on SE schemes. Moreover, since our attack exploits both the `access and volume pattern`, countermeasures must mitigate both leakage patterns. The former can be mitigated by padding the server result, while the latter may be handled using volume-hiding techniques. However, these approaches may come with impractical side effects. Padding the server response requires more work on the client-side to filter out the false positives. This padding can cause storage and reading problems because the user has to wait for the program to filter out the correct results. The volume-hiding technique [19] may easily yield significant storage overhead and could therefore not be practical in reality. Luckily, Patel et al. [29] illustrated how to reduce this side effect whilst mitigating the attack.

It is possible to mitigate our attack theoretically by using a combination of padding and volume hiding. We tested the VAL-attack's performance with padding, volume hiding and further a combination, but we did not examine by obfuscation due to hardware limitations.

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series,
and is available online at 10.1007/978-3-031-17140-6_32

We padded the server data using the technique described by Cash et al. [8]. Each query returned a multiplication of 500 server files, so if the original query returned 600 files, the server now returned 1,000. Padding is done by adding documents to the server response that to done contain the underlying keyword. These documents can then later be filtered by the client, but will obfuscate the client's observation. We took the naïve approach from Kamara et al. [19] for volume hiding, where we padded each document to the same volume. By adding empty bytes to a document, it will grow in size. If done properly, all files will eventually have the same size that can not be distinguished from the actual size.

We ran the countermeasure experiments on the Enron and the Lucene dataset. We did not perform the test on the Wikipedia dataset, but we can predict that the countermeasures may affect the attack performance. We predict that a single countermeasure will not entirely reduce the attack effectiveness, but a combination may do.

Because of the exploitation of the two leakage patterns, we see in Table 2 that our attack can still recover files and underlying keywords against only a single countermeasure. Under a combination of padding and volume hiding, our attack cannot reveal any leaked file or keyword.

Table 2: Performance of VAL-Attack with countermeasures

Dataset		Enron			Lucene		
Counter-measure		Padding	Volume Hiding	Padding & Vol. Hiding	Padding	Volume Hiding	Padding & Vol. Hiding
Files	0.1%	25 (83.7%)	27 (89.5%)	0 (0%)	45 (88.9%)	10 (28.4%)	0 (0%)
	0.5%	103 (68.4%)	137 (90.7%)	0 (0%)	191 (75.3%)	95 (37.4%)	0 (0%)
	1%	208 (69.0%)	274 (90.9%)	0 (0%)	381 (75.3%)	147 (28.9%)	0 (0%)
	5%	1,114 (74.0%)	1,365 (90.7%)	0 (0%)	2332 (92.0%)	2452 (96.8%)	0 (0%)
	10%	1,910 (63,4%)	2,736 (90.9%)	0 (0%)	4,073 (80.4%)	4,891 (96.5%)	0 (0%)
	30%	5,358 (59.0%)	8,219 (91.0%)	0 (0%)	10,343 (68.0%)	-[2]	0 (0%)
Queries	0.1%	94 (10.4%)	172 (14.8%)	0 (0%)	377 (27.7%)	153 (10.6%)	0 (0%)
	0.5%	433 (18.1%)	1,059 (43.3%)	0 (0%)	724 (25.3%)	663 (22.8%)	0 (0%)
	1%	414 (12.8%)	1,836 (56.3%)	0 (0%)	556 (15.3%)	748 (20.5%)	0 (0%)
	5%	53 (1.1%)	4,290 (89.9%)	0 (0%)	87 (1.8%)	4,659 (95.2%)	0 (0%)
	10%	11 (0.2%)	4,890 (98.4%)	0 (0%)	33 (0.7%)	4,872 (97.6%)	0 (0%)
	30%	1 (0.0%)	4,993 (99.9%)	0 (0%)	10 (0.2%)	-[2]	0 (0%)

[2] Did not run due to hardware limitations

Table 2 is read as follows: The number below the countermeasure is the exact number of retrieved files or queries, with the relative percentage between brackets. So for 0.1% leakage under the padding countermeasure, we revealed, on average, 25 files, which was 83.7% of the leaked files. Each experiment ran 20 times. Due to runtime and hardware limitations, we did not run the experiment with 30% leakage on the Lucene dataset. However, since we have the results for 10% leakage and the results for the Enron dataset, we can predict the outcome for 30%. Similar to the Enron dataset, the recovered data in Lucene increases as the leakage percentage grows. Therefore, we predict that 30% leakage results in the Lucene dataset is a bit higher than the 10% leakage.

5.4 Discussion on Experiments

We chose specific parameters in the experiments and only compared our attack with two popular attacks [4, 28]. We give more discussions below.

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

Parameters. We used 5,000 high selectivity keywords, i.e. keywords that occur the most in the dataset. This number is chosen because a practical SE application will probably not have just a few search terms in a real-world scenario. Other attacks [4, 8, 18] have experimented with only 150 query tokens and 500 keywords, and we argue that this may not be realistic. Our attack is able to recover almost all underlying keywords for an experiment with 500 keywords because the number of files is still equal, but a slight variation in keyword occurrence.

We cut the number of Wikipedia files to 50,000. We did this to better present the comparison with the Enron and Lucene datasets. The attack may also take longer to run when all Wikipedia files are considered. The results will also differ as the number of files leaked increases similarly. The percentage of files recovered will probably be the same because of keyword distribution among the files.

If we ran the experiments with a higher leakage percentage, the attack would eventually recover more files, as more are available, but we would not recover more keywords. As with 30% leakage, we see that we have recovered all 5,000 keywords.

Our attack performs without false positives. And we did so because they would not improve the performance, and an attacker cannot better understand the data if he cannot rely on it. If we allowed the attack to return false positives, we would have 5,000 matches for underlying keywords, of which not all are correct. The attack performance will not change since we will only measure the correct matches, which we already did.

Attack comparison. In Fig. 5a, we only compared our attack with the LEAP attack rather than the Subgraph$_{\mathrm{VL}}$ attack. We did so because the latter does not reveal encrypted files and thus cannot be compared. If we choose to compare the attack to ours, we would have to rebuild their attack using their strategy, which is out of the scope of this work.

We used the Enron dataset to compare the VAL-Attack to the LEAP and the Subgraph$_{\mathrm{VL}}$. In their work [4, 28], they used the Enron dataset to show their performance. If we used the Lucene or Wikipedia dataset instead to present the comparison, we would have no foundation in the literature to support our claim. A comparison of all the datasets would still show that our attack surpasses the attacks since, in theory, we exploit more.

We discussed other attacks, like the IKK and the Count attack, but we did not compare their performance with ours. While these attacks exploit the same leakage, we could still consider them. However, since LEAP is considered the most state-of-the-art attack and it has already been compared with the other attacks in [28], we thus only have to compare the LEAP attack here. Accordingly, a comparison with all attacks would not affect the results and conclusion of this paper.

6 Related Work

The Count attack [8] uses the number of files returned for the query as their matching technique; The SubgraphVL [4] matches keywords based on unique document volumes as if it is the response pattern, and the LEAP attack [28] uses techniques from previous attacks to match leaked documents and keywords with high accuracy. Besides the attacks that exploit similar leakage to our proposed attack, we may also review those attacks that do not. An attack that leverages similar documents as auxiliary knowledge, called Shadow Nemesis, was proposed by Pouliot et al. [31]. They created a weighted graph matching problem in the attack and solved it using Path or Umeyama. Damie et al. [14] presented the Score attack, requiring similar documents, and they matched based on the frequency of keywords in the server and auxiliary documents. Both attacks use co-occurrence matrices to reveal underlying keywords. The Search attack by Liu et al. [25] matches based on the search pattern, i.e. the frequency pattern of queries sent to the

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

server. Table 3 briefly compares the attacks based on leakage, auxiliary knowledge, false positives and exploiting techniques. The reviewed attacks described above are not mainly relevant to our proposed attack; thus, we did not put them in the comparison in Section 5.

Table 3: Comparison on Different Attacks. The lower part are those passive attacks with preknown data compared with VAL-Attack. *Documents* in the auxiliary data column refers to leaked document knowledge, *queries* refers to leaked underlying keywords for query tokens, and *similar* refers to the use of similar documents instead of leaked documents.

Attack	Leakage	Auxiliary data	False positives	Exploited information
IKK [18]	Access pattern	Documents, queries	✓	Co-occurrence
Shadow Nemesis [31]	Access pattern	Similar	✓	Co-occurrence
Score [14]	Access pattern	Similar, queries	✓	Co-occurrence
Search14 [25]	Search pattern	Search frequency	✓	Query frequency
ZKP [37] (active)	Access pattern	All keywords	✗	-
Count [8]	Access pattern	Documents	✓	Co-occurrence, length
Subgraph$_{\text{VL}}$ [4]	Volume pattern	Documents	✓	Volume, length
LEAP [28]	Access pattern	Documents	✗	Co-occurrence, length
VAL-Attack	Access, volume pattern	Documents	✗	Volume, length, co-occurrence

7 Conclusion

We proposed the VAL-attack to improve the matching technique from the LEAP attack, leveraging the leakage from the `access pattern` and the `volume pattern` which is a combination that has not been exploited before. We showed that our attack provides excellent performance, and we compared it to the LEAP attack and the subgraph$_{\text{VL}}$ attack. The number of matched files is with more remarkable improvement than the number of queries recovered compared to the LEAP attack. The attack recovers around 98% of the leaked documents and above 90% for query recovery with very low leakage. Since the proposed attack uses both the document size and the response per query, it requires strong (and combined) countermeasures and thus, is more harmful than existing attacks.

Acknowledgments. We would like to thank the anonymous reviewers for their helpful comments. Kaitai Liang is supported by European Union’s Horizon 2020 research and innovation programme under grant agreement No. 952697 (ASSURED) and No. 101021727 (IRIS). Jianting Ning was supported in part by the National Natural Science Foundation of China under Grant 61972094 and Grant 62032005.

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series,
and is available online at 10.1007/978-3-031-17140-6_32

References

1. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 205–222. Springer (2005). https://doi.org/10.1007/11535218_13
2. Apache: Mail archieves of lucene (1999), https://mail-archives.apache.org/mod_mbox/#lucene
3. Bird, S., Klein, E., Loper, E.: Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc (2009)
4. Blackstone, L., Kamara, S., Moataz, T.: Revisiting leakage abuse attacks. In: NDSS 2020. The Internet Society (2020). https://doi.org/10.14722/ndss.2020.23103
5. Boneh, D., Crescenzo, G.D., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer (2004). https://doi.org/10.1007/978-3-540-24676-3_30
6. Bost, R.: $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 1143–1154. ACM (2016). https://doi.org/10.1145/2976749.2978303
7. Bost, R., Minaud, B., Ohrimenko, O.: Forward and backward private searchable encryption from constrained cryptographic primitives. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1465–1482. ACM (2017). https://doi.org/10.1145/3133956.3133980
8. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 668–679. ACM (2015). https://doi.org/10.1145/2810103.2813700
9. Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., Steiner, M.: Dynamic searchable encryption in very-large databases: Data structures and implementation. In: NDSS 2014. The Internet Society (2014). https://doi.org/10.14722/ndss.2014.23264
10. Cash, D., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 353–373. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_20
11. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 577–594. Springer (2010). https://doi.org/10.1007/978-3-642-17373-8_33
12. Chen, G., Lai, T., Reiter, M.K., Zhang, Y.: Differentially private access patterns for searchable symmetric encryption. In: IEEE INFOCOM 2018. pp. 810–818. IEEE (2018). https://doi.org/10.1109/INFOCOM.2018.8486381
13. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) ACM CCS 2006. pp. 79–88. ACM (2006). https://doi.org/10.1145/1180405.1180417
14. Damie, M., Hahn, F., Peter, A.: A highly accurate query-recovery attack against searchable encryption using non-indexed documents. In: Bailey, M., Greenstadt, R. (eds.) USENIX 2021. pp. 143–160. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/damie
15. Demertzis, I., Papamanthou, C.: Fast searchable encryption with tunable locality. In: Salihoglu, S., Zhou, W., Chirkova, R., Yang, J., Suciu, D. (eds.) ACM SIGMOD 2017. pp. 1053–1067. ACM (2017). https://doi.org/10.1145/3035918.3064057

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

16. Gui, Z., Paterson, K.G., Patranabis, S.: Rethinking searchable symmetric encryption. Cryptology ePrint Archive, Paper 2021/879 (2021), https://eprint.iacr.org/2021/879
17. He, W., Akhawe, D., Jain, S., Shi, E., Song, D.X.: Shadowcrypt: Encrypted web applications for everyone. In: Ahn, G., Yung, M., Li, N. (eds.) ACM CCS 2014. pp. 1028–1039. ACM (2014). https://doi.org/10.1145/2660267.2660326
18. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: NDSS 2012. The Internet Society (2012), https://www.ndss-symposium.org/ndss2012/access-pattern-disclosure-searchable-encryption-ramification-attack-and-mitigation
19. Kamara, S., Moataz, T.: Computationally volume-hiding structured encryption. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 183–213. Springer (2019). https://doi.org/10.1007/978-3-030-17656-3_7
20. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A. (ed.) FC 2013. LNCS, vol. 7859, pp. 258–274. Springer (2013). https://doi.org/10.1007/978-3-642-39884-1_22
21. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012. pp. 965–976. ACM (2012). https://doi.org/10.1145/2382196.2382298
22. Kortekaas, Y.: Access pattern hiding aggregation over encrypted databases (October 2020), http://essay.utwente.nl/83874/
23. Lau, B., Chung, S.P., Song, C., Jang, Y., Lee, W., Boldyreva, A.: Mimesis aegis: A mimicry privacy shield-a system’s approach to data privacy on public cloud. In: Fu, K., Jung, J. (eds.) USENIX 2014. pp. 33–48. USENIX Association (2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/lau
24. Li, J., Niu, X., Sun, J.S.: A practical searchable symmetric encryption scheme for smart grid data. In: IEEE ICC 2019. pp. 1–6. IEEE (2019). https://doi.org/10.1109/ICC.2019.8761599
25. Liu, C., Zhu, L., Wang, M., Tan, Y.: Search pattern leakage in searchable encryption: Attacks and new construction. Information Sciences **265**, 176–188 (2014). https://doi.org/10.1016/j.ins.2013.11.021
26. Ma, Q., Zhang, J., Peng, Y., Zhang, W., Qiao, D.: SE-ORAM: A storage-efficient oblivious RAM for privacy-preserving access to cloud storage. In: Qiu, M., Tao, L., Niu, J. (eds.) IEEE CSCloud 2016. pp. 20–25. IEEE Computer Society (2016). https://doi.org/10.1109/CSCloud.2016.24
27. Minaud, B., Reichle, M.: Dynamic local searchable symmetric encryption. arXiv preprint (2022), https://arxiv.org/abs/2201.05006
28. Ning, J., Huang, X., Poh, G.S., Yuan, J., Li, Y., Weng, J., Deng, R.H.: LEAP: leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2307–2320. ACM (2021). https://doi.org/10.1145/3460120.3484540
29. Patel, S., Persiano, G., Yeo, K., Yung, M.: Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM SIGSAC 2019. pp. 79–93. ACM (2019). https://doi.org/10.1145/3319535.3354213
30. Porter, M.F.: An algorithm for suffix stripping. Program **40**, 211–218 (1980)
31. Pouliot, D., Wright, C.V.: The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM SIGSAC 2016. pp. 1341–1352. ACM (2016). https://doi.org/10.1145/2976749.2978401
32. Shang, Z., Oya, S., Peter, A., Kerschbaum, F.: Obfuscated access and search patterns in searchable encryption. In: NDSS 2021. The Internet Society (2021),

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

https://www.ndss-symposium.org/ndss-paper/obfuscated-access-and-search-patterns-in-searchable-encryption/

33. Shapiro, D.: Convert wikipedia database dumps into plaintext files (2021), https://github.com/daveshap/PlainTextWikipedia
34. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE S&P 2000. pp. 44–55. IEEE (2000). https://doi.org/10.1109/SECPRI.2000.848445
35. William W. Cohen, MLD, C.: Enron email datasets (2015), https://www.cs.cmu.edu/~enron/
36. Zhang, R., Imai, H.: Combining public key encryption with keyword search and public key encryption. IEICE Trans. Inf. Syst. **92-D**(5), 888–896 (2009). https://doi.org/10.1587/transinf.E92.D.888
37. Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: Holz, T., Savage, S. (eds.) USENIX 2016. pp. 707–720. USENIX Association (2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/zhang
38. Zheng, Q., Xu, S., Ateniese, G.: VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In: IEEE INFOCOM 2014. pp. 522–530. IEEE (2014). https://doi.org/10.1109/INFOCOM.2014.6847976

A Examples of Matrices

Figure 6: Matrix A and M' Example

$$\begin{array}{c|cccc} A & d_1 & d_2 & \cdots & d_{n'} \\ \hline w_1 & 1 & 1 & \cdots & 1 \\ w_2 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m'} & 1 & 0 & \cdots & 1 \end{array} \qquad \begin{array}{c|cccc} M' & d_1 & d_2 & \cdots & d_{n'} \\ \hline d_1 & 5 & 2 & \cdots & 3 \\ d_2 & 2 & 6 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{n'} & 3 & 0 & \cdots & 10 \end{array}$$

Figure 7: Matrix B and M Example

$$\begin{array}{c|cccc} B & ed_1 & ed_2 & \cdots & ed_n \\ \hline q_1 & 0 & 1 & \cdots & 1 \\ q_2 & 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_m & 1 & 1 & \cdots & 0 \end{array} \qquad \begin{array}{c|cccc} M & ed_1 & ed_2 & \cdots & ed_n \\ \hline ed_1 & 4 & 3 & \cdots & 1 \\ ed_2 & 3 & 9 & \cdots & 2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ ed_n & 1 & 2 & \cdots & 9 \end{array}$$

Figure 8: Matrix A_c and B_c Example

$$\begin{array}{c|cccc} B_c & ed_3 & ed_2 & \cdots & ed_t \\ \hline q_1 & 1 & 0 & \cdots & 1 \\ q_2 & 1 & 1 & \cdots & 0 \\ q_3 & 1 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_m & 0 & 1 & \cdots & 0 \end{array} \qquad \begin{array}{c|cccc} A_c & d_1 & d_2 & \cdots & d_t \\ \hline w_1 & 1 & 0 & \cdots & 1 \\ w_2 & 1 & 1 & \cdots & 0 \\ w_3 & 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_m & 1 & 1 & \cdots & 0 \end{array}$$

Figure 9: Matrix A_r and B_r Example

$$\begin{array}{c|cccc} B_r & ed_1 & ed_2 & \cdots & ed_n \\ \hline q_3 & 0 & 0 & \cdots & 1 \\ q_5 & 1 & 1 & \cdots & 0 \\ q_2 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_t & 1 & 1 & \cdots & 0 \end{array} \qquad \begin{array}{c|cccc} A_r & d_1 & d_2 & \cdots & d_{n'} \\ \hline w_1 & 1 & 0 & \cdots & 1 \\ w_2 & 0 & 0 & \cdots & 1 \\ w_3 & 1 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_t & 1 & 1 & \cdots & 0 \end{array}$$

“This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science book series, and is available online at 10.1007/978-3-031-17140-6_32

Figure 10: An Example of Extended Matrix A

$$\begin{array}{cc} A & \begin{array}{cccc} d_1 & d_2 & \cdots & d_{n'} \end{array} \\ \begin{array}{c} w_1 \\ w_2 \\ \vdots \\ w_{m'} \\ w_{m'+1} \\ \vdots \\ w_m \end{array} & \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \end{array}$$

B VAL-Attack Algorithm

Algorithm 4 VAL-Attack

```
Input: $A\ (m' \times n'), B\ (m \times n),\ M'\ (n' \times n'), M\ (n \times n)$
$C = R \leftarrow \{\}$                                    ▷ Initialization
$A \leftarrow A$ where rows extended with 0's (m x n')
$\text{vector}_A = \text{vector}_B \leftarrow [\,]$          ▷ Match documents with unique #keywords
for $j \in [n]$ do
    $\text{vector}_B[j] \leftarrow$ sum of column $B_j$
for $j' \in [n']$ do
    $\text{vector}_A[j'] \leftarrow$ sum of column $A_{j'}$
for $\text{vector}_{B_j} \in \text{vector}_B$ that is unique do
    if $\text{vector}_{A_{j'}} == \text{vector}_{B_j}$ then
        $C[ed_j] \leftarrow d_{j'}$
$C \leftarrow C \cup \text{matchByVolume}(R, A, B)$          ▷ Match documents with unique volume
$C \leftarrow C \cup \text{coOccurrence}(C, M, M', A, B)$   ▷ Match docs with co-occurrence
$C \leftarrow C \cup \text{matchByVolume}(R, A, B)$
while $R$ or $C$ is increasing do
    $R \leftarrow R \cup \text{matchKeywords}(C, A, B)$       ▷ Match keywords in matched docs
    $B_r \leftarrow$ R rows of $B$                           ▷ Match documents with unique keyword order
    $A_r \leftarrow$ R rows of $A$
    for $\text{column}_j \in B_r$ that is unique do
        if $\text{column}_{j'} \in A_r == \text{column}_j$ then
            $C[ed_j] \leftarrow d_{j'}$
    $C \leftarrow C \cup \text{matchByVolume}(R, A, B)$
    row $B_j \leftarrow 0$ if $q_j \in R$                    ▷ Match documents with unique #keywords
    row $A_{j'} \leftarrow 0$ if $k_{j'} \in R$
    for $j \in [n]$ where $ed_j \notin C$ do
        $\text{vector}_B[j] \leftarrow$ sum of column $B_j$
    for $j' \in [n']$ where $d_{j'} \notin C$ do
        $\text{vector}_A[j'] \leftarrow$ sum of column $A_{j'}$
    for $\text{vector}_{B_j} \in \text{vector}_B$ that is unique and $ed_j \notin C$ do
        if $\text{vector}_{A'_j} == \text{vector}_{B_j}$ and $d_{j'} \notin C$ then
            $C[ed_j] \leftarrow d_{j'}$
    $C \leftarrow C \cup \text{coOccurrence}(C, M, M', A, B)$   ▷ Match docs with co-occurrence
return $R$, $C$
```