LEAP: Leakage-Abuse Attack on Efficiently Deployable, Efficiently Searchable Encryption with Partially Known Dataset

Jianting Ning*
Fujian Normal University &
Singapore Management University
Fuzhou, China
jtning88@gmail.com

Xinyi Huang†
Fujian Normal University
Fuzhou, China
xyhuang81@gmail.com

Geong Sen Poh
S-Lab for Advanced Intelligence,
Nanyang Technological University
Singapore
geongsen.poh@ntu.edu.sg

Jiaming Yuan
University of Oregon
Oregon, United States
jiamingy@uoregon.edu

Yingjiu Li
University of Oregon
Oregon, United States
yingjiul@uoregon.edu

Jian Weng
Jinan University
Guangzhou, China
cryptjweng@gmail.com

Robert H. Deng
Singapore Management University
Singapore
robertdeng@smu.edu.sg

ABSTRACT

Searchable Encryption (SE) enables private queries on encrypted documents. Most existing SE schemes focus on constructing industrial-ready, practical solutions at the expense of information leakages that are considered acceptable. In particular, ShadowCrypt utilizes a cryptographic approach named "efficiently deployable, efficiently searchable encryption" (EDESE) that reveals the encrypted dataset and the query tokens among other information. However, recent attacks showed that such leakages can be exploited to (partially) recover the underlying keywords of query tokens under certain assumptions on the attacker's background knowledge.

We continue this line of work by presenting LEAP, a new leakage-abuse attack on EDESE schemes that can accurately recover the underlying keywords of query tokens based on partially known documents and the L2 leakage as per defined by Cash *et al.* (CCS '15). As an auxiliary function, our attack supports *document recovery* in the similar setting. To the best of our knowledge, this is the first attack on EDESE schemes that achieves *keyword recovery* and document recovery without error based on partially known documents and L2 leakage. We conduct extensive experiments to demonstrate the effectiveness of our attack by varying levels of attacker's background knowledge.

*The first author is also with the State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093).
†Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484540.

CCS CONCEPTS

• **Security and privacy** → **Management and querying of encrypted data**; **Cryptanalysis and other attacks**.

KEYWORDS

Searchable encryption; leakage; attack

ACM Reference Format:
Jianting Ning, Xinyi Huang, Geong Sen Poh, Jiaming Yuan, Yingjiu Li, Jian Weng, and Robert H. Deng. 2021. LEAP: Leakage-Abuse Attack on Efficiently Deployable, Efficiently Searchable Encryption with Partially Known Dataset. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3460120.3484540.

1 INTRODUCTION

Encrypted cloud storage systems have been developed to alleviate the privacy concerns of organisations that outsource their sensitive data to a third-party storage provider. Searchable encryption (SE) is one of the key solutions that attempt to preserve retrievability of encrypted data, without revealing the queried information to the storage provider. Since the seminal work by Song *et al.* [40], many practical SE schemes have been proposed [4, 6, 7, 9, 12, 25, 33, 41].

In order to provide efficient query on encrypted data stored on a remote cloud server, these practical SE schemes allow certain leakages of information that are deemed acceptable by users. Cash *et al.* [5] characterized the leakage profiles of SE schemes in the literature and in-the-wild SE products by defining a series of leakage levels L1-L4. L1 leakage, consisting of the *query-revealed occurrence pattern*, has the least amount of leakage. L2 leakage stands for the leakage of *fully-revealed occurrence pattern*, which leaks more information than L1 but less than L3 and L4. Due to the high efficiency of SE schemes with L2 leakage, they have been incorporated in a number of operational prototypes and products. ShadowCrypt [20] supports end-to-end encryption and SE with L2 leakage for web applications, such as Twitter, Facebook and Gmail. Specifically, ShadowCrypt interposes itself between a human user and the user interface of a web application. To keep the user's ability to efficiently search the stored documents, ShadowCrypt employs a type of SE with L2 leakage called *efficiently deployable, efficiently searchable encryption* (EDESE). In a typical implementation of an EDESE scheme, as used in ShadowCrypt, a list of encrypted keywords (here after referred to as *query tokens*) is attached to an encrypted document. Each query token q is calculated as a pseudorandom function F of a keyword w (keyed with a secret key k). To search for a keyword w, ShadowCrypt intercepts the search request and replaces the keyword w with the corresponding query token $q = F_k(w)$. EDESE schemes have several compelling advantages, including lower startup costs as compared to other types of SE and allow encryption of communications to be performed immediately without changing providers or losing familiar application user interfaces. On the other hand, EDESE schemes suffer from L2 leakage which could be exploited by an attacker. As the encrypted documents and the corresponding query tokens are stored on the server side, an adversarial server could obtain the relationship between each query token and each encrypted document. Particularly, whether a query token is contained in an encrypted document is leaked to the server. During the rest of the paper, we will use EDESE and SE with L2 leakage interchangeably unless otherwise stated.

It has been shown in recent years that the leakages of SE schemes can be exploited to recover the underlying keywords of query tokens, given full or partial background knowledge about the documents or the keywords contained in the documents. Islam *et al.* [21] initiated the investigation through empirical analysis on the security of SE and demonstrated that the underlying keywords of queries can be recovered if given (almost) all the documents. Following from this, Cash *et al.* [5] (CGPR) proposed an improved attack that successfully recovers query keywords using less prior knowledge about the (plaintext) documents of the user and L1 leakage. Pouliot and Wright [39] later proposed new inference attacks on EDESE schemes utilizing L2 leakage, one is based on the Umeyama's algorithm [42] (PW-U), and the other is based on the PATH algorithm [43] (PW-P). Independent from the above *passive* attacks, Cash *et al.* [5] also introduced a new type of attack in which an attacker can induce a user to insert chosen documents, which is essentially an *active* attack. A new active attack called *file-injection attack* was later proposed by Zhang *et al.* [44], which injects deliberately chosen documents into the document set of the user. Compared with active attacks, passive attacks require weaker assumption since the attacker only needs to observe the leakages of a SE scheme and hence is easier to launch. This paper focuses on passive attacks.

For passive attacks, a practical assumption could be that it is unlikely an attacker can obtain all the documents of a target user. On the other hand, it seems too restrictive to assume that the attacker knows no plaintext document of the user at all. For instance, a storage provider can easily learn the nature of a user's business. The provider may then construct the common keywords and gather common documents reflecting the business domain (e.g., finance, healthcare). It is also over optimistic from a security standpoint to assume the attacker has no way to learn partial information. Thus, a more realistic assumption is to assume that the attacker could obtain a partial set of the documents for a target user. As noted in [5], "assuming knowledge of no documents is a step too far", and an attacker may know that one or more widely-circulated emails are stored in a user's repository.

To date, there are only a few works that focus on such practical scenario where the attacker has only partial knowledge of a target user's document set. However, the attack results reported in these works may contain false positives due to the lack of knowledge of the missing documents. In particular, the PW-U attack and the PW-P attack (on EDESE schemes) proposed by Pouliot and Wright [39] are two types of attacks that work with partial knowledge of a dataset. Both attacks result in false positives. As the experimental results shown in Section 5, when given 10% of documents of the dataset with 4,991 keyword universe, the PW-U attack returns 4,991 (query token, keyword) mappings but only 38 keywords are correctly mapped, and the PW-P attack returns 4,991 (query token, keyword) mappings but only 1,638 mappings are correct. The main idea of the PW-U attack and the PW-P attack is to reduce the problem of finding (query token, keyword) mapping to well-known combinatorial optimization problems based on graph matching. However, due to the nature of the combinatorial optimization problems based on graph matching, the recovered (query token, keyword) mappings may contain false positives when the attacker's knowledge about a target user's document set is not complete.

The CGPR attack proposed by Cash *et al.* [5] is another type of attack in such partial knowledge setting, which also results in false positives. The difficulty of accurately recovering the underlying keywords of query tokens lies in the information loss induced by missing documents. Due to the missing documents, the attacker cannot simply recover a query token q by finding a keyword w with a unique $\mathbf{count}(w)$ such that $\mathbf{count}(q) = \mathbf{count}(w)$, where $\mathbf{count}(w)$ denotes the number of *known documents* containing w and $\mathbf{count}(q)$ denotes the number of encrypted documents containing q. This is because the document set corresponding to $\mathbf{count}(w)$ is a subset of the full document set; consequently, $\mathbf{count}(w)$ may be less than the number of documents from the full document set containing w. As a result, the attack under the partial knowledge setting in [5] utilizes a guessing strategy to prepare a candidate keyword set for a query token, which serves as the basis for later pruning. This is the reason why the attack usually outputs (query token, keyword) mappings with (high) false positives.

Intuitively, the criterion to measure the effectiveness of an inference attack on SE scheme is how many (query token, keyword) mappings can be *accurately* recovered, rather than how many (query token, keyword) mappings are output by the attack (which may contain false positives). The following question arises naturally:

Can a passive adversarial server reveal query tokens (i. e., discovering the underlying keywords of query tokens) accurately with only partial knowledge of the user's document set?

1.1 Our Contributions

In this paper, we attempt to address the above problem by presenting a new leakage-abuse attack on EDESE schemes, named LEAP (Leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset). Different from the guessing strategy in [5] and the graph matching approach in [39], we introduce a new approach which reveals query tokens accurately with only partial knowledge of the dataset.

In a nutshell, we first accurately recover certain (encrypted document, document) mappings. This step relies on two methods (see **Method 1** and **Method 2** described in Section 4.2) and the observation that the $m' \times n'$ document-keyword matrix $\mathbf{A}'$ (derived from known documents) can be extended to a new $m \times n'$ document-keyword matrix $\mathbf{A}''$, where n' is the number of leaked documents, m' is the number of keywords contained in the leaked documents and m is the number of keywords in the keyword universe. With the recovered (encrypted document, document) mappings, we can then recover certain (query token, keyword) mappings without error based on **Method 3** described in Section 4.2. Next, we use a recursive mechanism to recover more (query token, keyword) mappings. In particular, the recovered (query token, keyword) mappings are used to discover new (encrypted document, document) mappings which in turn are used to discover new (query token, keyword) mappings. This recursive discovery process is made possible based on a novel matrix row/column mapping technique we develop, which utilizes the leakage of EDESE schemes and the partial knowledge of a user's document set.

LEAP achieves zero false positives in breaking query token privacy in the sense that all (query token, keyword) mappings output by LEAP are correct. As an auxiliary function, it breaks document privacy without false positives, i.e., all (encrypted document, document) mappings output by LEAP are correct. As far as we know, this is the first attack utilizing only partial knowledge of the document set and the L2 leakage of EDESE schemes, yet is capable of recovering the user's query tokens and the encrypted documents *accurately* (i.e., without false positives).

We conduct extensive experiments to demonstrate the effectiveness of LEAP as compared to the PW-U attack and the PW-P attack. Given access to 10% of the dataset, LEAP accurately recovers 4,904 (query token, keyword) mappings out of 4,991 keywords, as compared to 1,638 in the PW-P attack and 38 in the PW-U attack. In the case where only 0.1% of the dataset is leaked, LEAP accurately recovers 132 (query token, keyword) mappings out of 1,144 keywords, as compared to 2 in the PW-P attack and 5 in the PW-U attack. The experimental results confirm that LEAP is devastating for the privacy of query tokens. LEAP reveals new risks of using EDESE schemes given a prior knowledge of the dataset. LEAP also highlights the importance of minimizing the leakage of a data storage or processing server.

2 PRELIMINARIES

2.1 Notation

Throughout this paper, we use d, w, ed, and q to denote a document, a keyword, an encrypted document, and a query token, respectively. We use d_i to denote a particular document$_i$, and use w_i, ed_i, and q_i similarly. Note that d (resp. w) is indexed independently from ed (resp. q). In other words, ed_i may not be the encryption of d_i, and q_i may not be the query token corresponding to w_i, even though they share the same subscript. In addition, we use (ed, d) to denote a mapping between an encrypted document and the corresponding plaintext document, and use (q, w) to denote a mapping between a query token and the corresponding keyword.

We define $[n] = \{1, 2, ..., n\}$ for $n \in \mathbb{N}$. For two vectors VA and VB of the same dimension, we define $VA = VB$ iff $VA[i] = VB[i]$ for all i. For an $m \times n$ matrix $\mathbf{T}$ where the (i, j)-th entry $T_{i,j}$ is either 1 or 0, let $column_j$ be the j-th column of $\mathbf{T}$ for $j \in [n]$, and row_i be the i-th row of $\mathbf{T}$ for $i \in [m]$. For column$_j$, let $T_{1,j}T_{2,j}...T_{m,j}$ be its *bit-string*; similarly, for row$_i$, let $T_{i,1}T_{i,2}...T_{i,n}$ be its bit-string. We say column$_j$ is *unique* if the bit-string of column$_j$ is unique among $\{\text{bit-string of column}_{j'}\}_{j' \in [n]}$; similarly, we say row$_i$ is *unique* if the bit-string of row$_i$ is unique among $\{\text{bit-string of row}_{i'}\}_{i' \in [m]}$. Let $column_j$*-sum* be the Hamming weight of the j-th column, i.e., $column_j$*-sum* equals $T_{1,j} + T_{2,j} + ... + T_{m,j}$; similarly, let row_i*-sum* be the Hamming weight of the i-th row. We take the following 5×6 matrix as an example:

$$\begin{array}{c c} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}, \tag{1}$$

the bit-string of column$_3$ is 10110, and the bit-string of row$_4$ is 011001. Column$_3$ is unique, and so are row$_2$, row$_3$, row$_5$. The column$_3$-sum is $1 + 0 + 1 + 1 + 0 = 3$, and the row$_2$-sum is $1 + 0 + 0 + 1 + 1 + 0 = 3$.

2.2 Background

We first give a general description of SE. In a SE scheme, a user encrypts her documents and uploads the encrypted documents to a (untrusted) server. Later, the user can issue a query containing a keyword (or a set of keywords) by generating and sending a query token to the server to retrieve the documents containing this keyword (or these keywords). Based on the query token, the server searches the stored encrypted documents and returns the encrypted documents (or the document identifiers) containing the queried keyword to the user.

The setting that we focus on is similar to the settings discussed in [5, 39] for EDESE schemes. In this setting, the keywords are encrypted with keyed pseudorandom function as the query tokens and appended to the encrypted documents stored on the server side. Similar to [5, 39], we assume the query of a keyword is processed as follows: the user first *deterministically* generates a query token from the keyword and sends a query request containing the query token to the server; with the query token, the server returns the encrypted documents which are attached with the query token in the query request. Since the encrypted documents and the attached query tokens are stored on the server, an adversarial server could obtain (1) the encrypted document universe and the query token universe, and (2) the relationship between each encrypted document and each query token, i.e., whether a query token is contained in an encrypted document. LEAP relies on such leakage. Similar to [5, 21, 39], only the "one-to-one" setting is considered for simplicity, where a query token corresponds to a single keyword. We leave the "one-to-many" setting where one query token may contain multiple keywords as our future work.

Let $\mathbf{F} = \{d_1, \ldots, d_n\}$ denote a set of (plaintext) documents of a target user. Each document d_i is represented by a set of keywords, $W_i = \{w_{i,1}, \ldots, w_{i,m_i}\}$, which can be extracted using an extraction algorithm. Let $\mathbf{W} = \{w_1, \dots, w_m\}$ denote the set of keywords appear in $\mathbf{F}$. The relation between $\mathbf{F}$ and $\mathbf{W}$ is encoded in a *document-keyword matrix* $\mathbf{A} = [A_{ij}]_{m\times n}$, where $A_{i,j}$ equals 1 iff document d_j contains keyword w_i, and 0 otherwise. The matrix is illustrated as follows:

$$\begin{array}{c} \\ w_1 \\ w_2 \\ \vdots \\ w_m \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & \cdots & d_n \end{array} \\ \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{pmatrix} \end{array}, \tag{2}$$

where document d_j is associated with column_j, and keyword w_i is associated with row_i. Column_j-sum of $\mathbf{A}$ captures the number of keywords that are contained in document d_j, and row_i-sum of $\mathbf{A}$ captures the number of documents that contain keyword w_i.

Let $\mathbf{E} = \{ed_1, ..., ed_n\}$ be the encrypted document set corresponding to $\mathbf{F}$, and let $\mathbf{Q} = \{q_1, ..., q_m\}$ be the query token set corresponding to $\mathbf{W}$. We further define an *encrypted document-query token matrix* $\mathbf{B} = [B_{ij}]_{m\times n}$ with entry $B_{i,j}$ equals 1 iff query token q_i is attached to encrypted document ed_j, and 0 otherwise. Matrix $\mathbf{B}$ is illustrated as follows:

$$\begin{array}{c} \\ q_1 \\ q_2 \\ \vdots \\ q_m \end{array} \begin{array}{c} \begin{array}{cccc} ed_1 & ed_2 & \cdots & ed_n \end{array} \\ \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,n} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ B_{m,1} & B_{m,2} & \cdots & B_{m,n} \end{pmatrix} \end{array}, \tag{3}$$

where ed_j is an encrypted document associated with column_j, and q_i is a query token associated with row_i. Column_j-sum of $\mathbf{B}$ captures the number of query tokens that are attached to encrypted document ed_j, and row_i-sum of $\mathbf{B}$ captures the number of encrypted documents where the sets of attached query tokens contain q_i.

If column_j of $\mathbf{B}$ matches $\text{column}_{j'}$ of $\mathbf{A}$, then we say ed_j is the encrypted version of $d_{j'}$, and we can thus obtain a mapping $(ed_j, d_{j'})$, where ed_j is the encrypted document corresponding to column_j of $\mathbf{B}$, and $d_{j'}$ is the document corresponding to $\text{column}_{j'}$ of $\mathbf{A}$; similarly, if row_i of $\mathbf{B}$ matches $\text{row}_{i'}$ of $\mathbf{A}$, then we say the underlying keyword of q_i is $w_{i'}$, and we can thus obtain a mapping $(q_i, w_{i'})$, where q_i is the query token corresponding to row_i of $\mathbf{B}$, and $w_{i'}$ is the keyword corresponding to $\text{row}_{i'}$ of $\mathbf{A}$.

The matrix representation above is similar to that of [5, 21], which generalises the inverted index used in most high efficiency SE schemes.

We then define an $n \times n$ *d-occurrence matrix* whose (i, j)-th entry captures the number of *keywords* that appear in both d_i and d_j. We also define an $n \times n$ *ed-occurrence matrix* whose (i, j)-th entry captures the number *of query tokens* that are attached to both ed_i and ed_j.

3 ATTACK MODEL

3.1 Attacker Type

As defined in [5, 21, 39], the attacker is an adversarial server who stores the encrypted documents and the corresponding query tokens. The attacker we consider in this paper is *passive* in the sense that it faithfully follows the EDESE schemes but attempts to learn more information than is allowed by examining the information it can observe. Intuitively, this type of attacker is weaker than the *active* attacker addressed in [44], which can trick a user into adding a document that is (deliberately) chosen by the attacker. In addition, the attacker has no access to any encryption or decryption oracles.

3.2 Attacker Knowledge

The knowledge of attacker includes the leakage of the EDESE schemes and the prior knowledge of a target user's documents.

For the leakage of the EDESE schemes, we consider the information leakage from the stored encrypted documents and the attached query tokens as described in Section 2. In particular, the attacker can utilize the information leaked by EDESE scheme to obtain the relationship between each encrypted document and each query token, i.e., which query token is attached to which encrypted document.

In terms of prior knowledge, we consider *partially-known document set*, which means that a subset of the (plaintext) documents of a target user is known to the attacker. For example, a set of widely-distributed emails may exist in the repository of a user and is known to the attacker, as articulated in [5]. The following is a scenario cited in [39] which is likely to happen in the real world. Suppose a user has a large corpus of documents stored on a service like Gmail, who decides to have all the documents encrypted with EDESE and uploaded to the server. Clearly, the server has perfect knowledge of the old plaintext corpus. Over time, new encrypted documents are uploaded to the server. In this scenario, the server has partial knowledge of the user's documents.

Unlike previous inference attacks, our attack does not require (1) any priori knowledge of the query requests, (2) any prior knowledge of the distribution of queries, and (3) any prior knowledge on the underlying keywords of any query tokens, i.e., the mapping between a (plaintext) keyword and the corresponding query token.

3.3 Objective of Attacker

The main objective of the attacker is *keyword recovery*, which is to recover the underlying keywords of a user's query tokens. Another objective is *document recovery*, which is to recover the relationship between known documents and encrypted documents.

4 LEAP

We now present LEAP, a new leakage-abuse attack against EDESE schemes with partial knowledge of a target user's document set.

4.1 Knowledge of Attacker

Let $\mathbf{F} = \{d_1, ..., d_n\}$ denote the full document set and $\mathbf{W} = \{w_1, ..., w_m\}$ denote the corresponding set of keywords. Let $\mathbf{F}' = \{d_{y_1}, ..., d_{y_{n'}}\}$ be the partial knowledge of the document set known to the attacker, and $\mathbf{W}' = \{w_{x_1}, ..., w_{x_{m'}}\}$ be the keyword set corresponding to $\mathbf{F}'$, where $\{y_1, ..., y_{n'}\} \subset [n]$ and $\{x_1, ..., x_{m'}\} \subset [m]$. Since each document consists of a set of keywords, $\mathbf{W}'$ can be easily derived from $\mathbf{F}'$ by the attacker.

With $\mathbf{F'}$ and $\mathbf{W'}$, the attacker can derive the following $m' \times n'$ document-keyword matrix $\mathbf{A'}$:

$$\begin{array}{c} \\ w_{x_1} \\ w_{x_2} \\ \vdots \\ w_{x_{m'}} \end{array} \begin{array}{c} \begin{array}{cccc} d_{y_1} & d_{y_2} & \cdots & d_{y_{n'}} \end{array} \\ \begin{pmatrix} A'_{1,1} & A'_{1,2} & \cdots & A'_{1,n'} \\ A'_{2,1} & A'_{2,2} & \cdots & A'_{2,n'} \\ \vdots & \vdots & \cdots & \vdots \\ A'_{m',1} & A'_{m',2} & \cdots & A'_{m',n'} \end{pmatrix} \end{array}, \tag{4}$$

where $A'_{i,j}$ is 1 if w_{x_i} is a keyword in document d_{y_j} for $i \in [m']$ and $j \in [n']$, and 0 otherwise.

In addition, the attacker obtains an $n' \times n'$ d-occurrence matrix $\mathbf{M'}$ as follows:

$$\begin{array}{c} \\ d_{y_1} \\ d_{y_2} \\ \vdots \\ d_{y_{n'}} \end{array} \begin{array}{c} \begin{array}{cccc} d_{y_1} & d_{y_2} & \cdots & d_{y_{n'}} \end{array} \\ \begin{pmatrix} M'_{1,1} & M'_{1,2} & \cdots & M'_{1,n'} \\ M'_{2,1} & M'_{2,2} & \cdots & M'_{2,n'} \\ \vdots & \vdots & \cdots & \vdots \\ M'_{n',1} & M'_{n',2} & \cdots & M'_{n',n'} \end{pmatrix} \end{array}, \tag{5}$$

where $M'_{i,j}$ is the number of keywords that appear in both d_{y_i} and d_{y_j} for $i,j \in [n']$.

Let $\mathbf{E} = \{ed_1, ..., ed_n\}$ be the encrypted document set of $\mathbf{F}$, and $\mathbf{Q} = \{q_1, ..., q_m\}$ be the query token set corresponding to $\mathbf{W}$. From the encrypted documents and the attached query tokens stored on the server, the attacker can derive the following $m \times n$ encrypted document-query token matrix $\mathbf{B}$:

$$\begin{array}{c} \\ q_1 \\ q_2 \\ \vdots \\ q_m \end{array} \begin{array}{c} \begin{array}{cccc} ed_1 & ed_2 & \cdots & ed_n \end{array} \\ \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,n} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ B_{m,1} & B_{m,2} & \cdots & B_{m,n} \end{pmatrix} \end{array}, \tag{6}$$

where $B_{i,j}$ is 1 if q_i is attached to ed_j for $i \in [m]$ and $j \in [n]$, and 0 otherwise.

In addition, the attacker can obtain the following $n \times n$ ed-occurrence matrix $\mathbf{M}$:

$$\begin{array}{c} \\ ed_1 \\ ed_2 \\ \vdots \\ ed_n \end{array} \begin{array}{c} \begin{array}{cccc} ed_1 & ed_2 & \cdots & ed_n \end{array} \\ \begin{pmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,n} \\ M_{2,1} & M_{2,2} & \cdots & M_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ M_{n,1} & M_{n,2} & \cdots & M_{n,n} \end{pmatrix} \end{array}, \tag{7}$$

where $M_{i,j}$ is the number of query tokens that are attached to both ed_i and ed_j for $i,j \in [n]$.

4.2 Technical Intuitions

Our main intuition is that by recursively finding and then sifting the row and column mappings between $\mathbf{A'}$ and $\mathbf{B}$, we can accurately recover the underlying keywords of the query tokens as well as the correspondence between the known documents and the encrypted documents. First observe that each encrypted document uniquely corresponds to a (plaintext) document. That is, there exists a subset $S_{col} \subset \{ed_1, ..., ed_n\}$ such that $\{\pi_1(d_{y_1}), ..., \pi_1(d_{y_{n'}})\} = S_{col}$, where π_1 is a function. Hence, for each column of $\mathbf{A'}$, there must exist a "matching" column in $\mathbf{B}$ [1]. Similarly, note that each query token uniquely corresponds to a keyword. That is, there exists a subset $S_{row} \subset \{q_1, ..., q_m\}$ such that $\{\pi_2(w_{x_1}), ..., \pi_2(w_{x_{m'}})\} = S_{row}$, where π_2 is a function. Hence, for each row of $\mathbf{A'}$, there exists a "matching" row in $\mathbf{B}$ [2]. Naturally, the goal of LEAP is reduced to finding the column mapping and row mapping between $\mathbf{B}$ and $\mathbf{A'}$.

Now recall the meaning of column-sum and row-sum of $\mathbf{B}$ and $\mathbf{A'}$ defined in Section 2. The column$_j$-sum of $\mathbf{B}$ captures the number of query tokens that are attached to encrypted document ed_j. The column$_j$-sum of $\mathbf{A'}$ captures the number of keywords that appear in document d_{y_j}. Clearly, one cannot simply match the rows between $\mathbf{B}$ and $\mathbf{A'}$ by finding unique row-sum mappings between them since $\mathbf{A'}$ has fewer columns than $\mathbf{B}$. For example, suppose that row$_4$-sum of $\mathbf{B}$ is z, and this value is unique among all row-sums of $\mathbf{B}$. Further assume that there exists a unique row$_5$-sum of $\mathbf{A'}$ equal to z. One cannot conclude that w_{x_5} is the underlying keyword of q_4. This is because the true value of row$_5$-sum may exceed z, since the missing documents may also contain w_{x_5}.

Instead, we map the columns between $\mathbf{B}$ and $\mathbf{A'}$ by finding unique column-sum mappings between them. From the encrypted documents and the attached query tokens stored on the server, we can derive m distinct query tokens. Since each query token uniquely corresponds to a keyword, we know that there are totally m keywords corresponding to the full document set (i.e., $\mathbf{F}$). Let $\{w_{x_{m'+1}}, ..., w_{x_{m''}}\} = \{w_1, ..., w_m\} - \{w_{x_1}, ..., w_{x_{m'}}\}$ be the keywords that do not appear in the partially-known document set $\mathbf{F'}$ (where $\mathbf{F'} = \{d_{y_1}, ..., d_{y_{n'}}\}$); in other words, $\{w_{x_{m'+1}}, ..., w_{x_{m''}}\}$ are unknown to the attacker. We can extend the $m' \times n'$ matrix $\mathbf{A'}$ to a new $m \times n'$ matrix $\mathbf{A''}$ by setting $\mathbf{A}''_{i,j} = 0$ for $i \in \{m'+1, ..., m''\}$ and $j \in \{1, ..., n'\}$ as follows:

$$\begin{array}{c} \\ w_{x_1} \\ w_{x_2} \\ \vdots \\ w_{x_{m'}} \\ w_{x_{m'+1}} \\ \vdots \\ w_{x_{m''}} \end{array} \begin{array}{c} \begin{array}{cccc} d_{y_1} & d_{y_2} & \cdots & d_{y_{n'}} \end{array} \\ \begin{pmatrix} A''_{1,1} & A''_{1,2} & \cdots & A''_{1,n'} \\ A''_{2,1} & A''_{2,2} & \cdots & A''_{2,n'} \\ \vdots & \vdots & \cdots & \vdots \\ A''_{m',1} & A''_{m',2} & \cdots & A''_{m',n'} \\ A''_{m'+1,1} = 0 & A''_{m'+1,2} = 0 & \cdots & A''_{m'+1,n'} = 0 \\ \vdots & \vdots & \cdots & \vdots \\ A''_{m'',1} = 0 & A''_{m'',2} = 0 & \cdots & A''_{m'',n'} = 0 \end{pmatrix} \end{array}, \tag{8}$$

Let d_{y_j} be a document associated with column$_j$ of $\mathbf{A''}$, w_{x_i} be a keyword associated with row$_i$ of $\mathbf{A''}$.

For the relationship of $\mathbf{A''}$ and $\mathbf{B}$, we have $\{\pi_1(d_{y_1}), ..., \pi_1(d_{y_{n'}})\} \subset \{ed_1, ..., ed_n\}$ and $\{\pi_2(w_{x_1}), ..., \pi_2(w_{x_{m''}})\} = \{q_1, ..., q_m\}$. The goal of LEAP is now reduced to the task of finding as many unique row mappings and unique column mappings as possible between $\mathbf{B}$ and $\mathbf{A''}$.

In more details, LEAP utilizes the following methods to find unique row mappings and unique column mappings between $\mathbf{B}$ and $\mathbf{A''}$.

[1] Here, "matching" denotes that for the document d corresponding to a particular column of $\mathbf{A'}$, there exists an encrypted document ed corresponding to a particular column in $\mathbf{B}$ such that $\pi_1(d) = ed$.

[2] Here, "matching" denotes that for the keyword w corresponding to a particular row of $\mathbf{A'}$, there exists a query token q corresponding to a particular row in $\mathbf{B}$ such that $\pi_2(w) = q$.

- **Method 1.** Since the number of rows in $\mathbf{A}''$ equals the number of rows in $\mathbf{B}$, we can find unique column-sum mappings between $\mathbf{B}$ and $\mathbf{A}''$ as follows: for each column$_j$-sum of $\mathbf{B}$ that is unique among $\{\text{column}_j\text{-sum of } \mathbf{B}\}_{j\in[n]}$, if we can find a column$_{j'}$-sum of $\mathbf{A}''$ which equals the column$_j$-sum of $\mathbf{B}$, then we can conclude that ed_j is the encrypted version of $d_{y_{j'}}$.
- **Method 2.** Given known column mappings, we employ $n \times n$ ed-occurrence matrix $\mathbf{M}$ and $n' \times n'$ d-occurrence matrix $\mathbf{M}'$ to find the column mappings between $\mathbf{B}$ and $\mathbf{A}''$ that cannot be mapped via unique column-sum as described in **Method 1**. The detailed algorithm of this method is shown in Algorithm 1. The intuition behind this algorithm is described as follows. For the relationship between $\mathbf{M}$ and $\mathbf{M}'$, $M_{i,j}$ equals $M'_{i',j'}$ if ed_i is the encrypted version of $d_{y_{i'}}$ and ed_j is the encrypted version of $d_{y_{j'}}$. For a known mapping $(ed_k, d_{y_{k'}})$ and a (unmapped) $d_{y_{j'}}$, we can obtain a new mapping $(ed_j, d_{y_{j'}})$ if there exists only one ed_j satisfying $M_{j,k} = M'_{j',k'}$ and column$_{j'}$-sum of $\mathbf{A}''$ equals column$_j$-sum of $\mathbf{B}$.
- **Method 3.** Given known (ed, d) mappings, this method aims to find (q, w) mappings. Without loss of generality, let $S_c = \{(ed_{j_1}, d_{y_{j'_1}}), \ldots, (ed_{j_t}, d_{y_{j'_t}})\}$ be the set of (ed, d) mappings that have been found, where $\{j_1, \ldots, j_t\} \subset [n]$ and $\{y_{j'_1}, \ldots, y_{j'_t}\} \subseteq [n']$. We define *$S_c$-column-mapped submatrix pair $(\mathbf{B}_c, \mathbf{A}''_c)$ from $(\mathbf{B}, \mathbf{A}'')$* as follows.

  Let $\mathbf{B}_c$ be a submatrix of $\mathbf{B}$ with a rearranged column order as:
$$\begin{array}{cc} & \begin{array}{cccc} ed_{j_1} & ed_{j_2} & \cdots & ed_{j_t} \end{array} \\ \begin{array}{c} q_1 \\ q_2 \\ \vdots \\ q_m \end{array} & \begin{pmatrix} B_{1,j_1} & B_{1,j_2} & \cdots & B_{1,j_t} \\ B_{2,j_1} & B_{2,j_2} & \cdots & B_{2,j_t} \\ \vdots & \vdots & \cdots & \vdots \\ B_{m,j_1} & B_{m,j_2} & \cdots & B_{m,j_t} \end{pmatrix} \end{array}. \tag{9}$$
  and let $\mathbf{A}''_c$ be a submatrix of $\mathbf{A}''$ with a rearranged column order as:
$$\begin{array}{cc} & \begin{array}{cccc} d_{y_{j'_1}} & d_{y_{j'_2}} & \cdots & d_{y_{j'_t}} \end{array} \\ \begin{array}{c} w_1 \\ w_2 \\ \vdots \\ w_m \end{array} & \begin{pmatrix} A''_{1,j'_1} & A''_{1,j'_2} & \cdots & A''_{1,j'_t} \\ A''_{2,j'_1} & A''_{2,j'_2} & \cdots & A''_{2,j'_t} \\ \vdots & \vdots & \cdots & \vdots \\ A''_{m,j'_1} & A''_{m,j'_2} & \cdots & A''_{m,j'_t} \end{pmatrix} \end{array}. \tag{10}$$
  Note that the columns in $\mathbf{B}_c$ are arranged according to the order of $(ed_{j_1}, ed_{j_2}, \ldots, ed_{j_t})$, while the columns in $\mathbf{A}''_c$ are arranged according to the order of $(d_{y_{j'_1}}, d_{y_{j'_2}}, \ldots, d_{y_{j'_t}})$.

  If any row$_i$ of $\mathbf{B}_c$ is unique among all rows of $\mathbf{B}_c$, then row$_i$ of $\mathbf{B}$ is unique among all rows of $\mathbf{B}$. The same applies to the case of $\mathbf{A}''_c$ and $\mathbf{A}''$. Hence, for each row$_i$ of $\mathbf{B}_c$ whose bit-string is unique among all rows of $\mathbf{B}_c$, if there exists a row$_{i'}$ of $\mathbf{A}''_c$ whose bit-string is the same as the bit-string of row$_i$ of $\mathbf{B}_c$, then we can conclude that the underlying keyword of q_i is $w_{i'}$.
- **Method 4.** This method is dual to **Method 3**. Given known (q, w) mappings, this method is used to find one or more (ed, d) mappings. Without loss of generality, let $S_r = \{(q_{i_1}, w_{x_{i'_1}}), \ldots, (q_{i_t}, w_{x_{i'_t}})\}$ be the set of (q, w) mappings that have been recovered, where $\{i_1, \ldots, i_t, i'_1, \ldots, i'_t\} \subseteq [m]$. We define *$S_r$-row-mapped submatrix pair $(\mathbf{B}_r, \mathbf{A}''_r)$ from $(\mathbf{B}, \mathbf{A}'')$* as follows:

  $\mathbf{B}_r$ is a submatrix of $\mathbf{B}$ with a rearranged row order as:
$$\begin{array}{cc} & \begin{array}{cccc} ed_1 & ed_2 & \cdots & ed_n \end{array} \\ \begin{array}{c} q_{i_1} \\ q_{i_2} \\ \vdots \\ q_{i_t} \end{array} & \begin{pmatrix} B_{i_1,1} & B_{i_1,2} & \cdots & B_{i_1,n} \\ B_{i_2,1} & B_{i_2,2} & \cdots & B_{i_2,n} \\ \vdots & \vdots & \cdots & \vdots \\ B_{i_t,1} & B_{i_t,2} & \cdots & B_{i_t,n} \end{pmatrix} \end{array}, \tag{11}$$
  and let $\mathbf{A}''_r$ be a submatrix of $\mathbf{A}''$ with a rearranged row order as:
$$\begin{array}{cc} & \begin{array}{cccc} d_{y_1} & d_{y_2} & \cdots & d_{y_{n'}} \end{array} \\ \begin{array}{c} w_{x_{i'_1}} \\ w_{x_{i'_2}} \\ \vdots \\ w_{x_{i'_t}} \end{array} & \begin{pmatrix} A''_{i'_1,1} & A''_{i'_1,2} & \cdots & A''_{i'_1,n'} \\ A''_{i'_2,1} & A''_{i'_2,2} & \cdots & A''_{i'_2,n'} \\ \vdots & \vdots & \cdots & \vdots \\ A''_{i'_t,1} & A''_{i'_t,2} & \cdots & A''_{i'_t,n'} \end{pmatrix} \end{array}, \tag{12}$$
  The rows in $\mathbf{B}_r$ are arranged according to the order of $(q_{i_1}, \ldots, q_{i_t})$, while the rows in $\mathbf{A}''_r$ are arranged according to the order of $(w_{x_{i'_1}}, \ldots, w_{x_{i'_t}})$.

  If any column$_j$ of $\mathbf{B}_r$ is unique among all columns of $\mathbf{B}_r$, then column$_j$ of $\mathbf{B}$ is unique among all columns of $\mathbf{B}$. The same applies to the case of $\mathbf{A}''_r$ and $\mathbf{A}''$. Hence, for each column$_j$ of $\mathbf{B}_r$ whose bit-string is unique among all columns of $\mathbf{B}_r$, if there exists a column$_{j'}$ of $\mathbf{A}''_r$ whose bit-string is the same as the bit-string of column$_j$ of $\mathbf{B}_r$, then we know that $d_{y_{j'}}$ is the plaintext of ed_j.
- **Method 5.** This method aims to find more column mappings. We use a vector VB_j (resp. vector $VA_{j'}$) to record the column-sum for each column$_j$ of $\mathbf{B}$ (resp. column$_{j'}$ of $\mathbf{A}''$) in each iteration. As the first step, VB_j (resp. $VA_{j'}$) records column$_j$-sum of $\mathbf{B}$ (resp. column$_{j'}$-sum of $\mathbf{A}''$) as its first element, while the rest of the elements are set to zero. Without loss of generality, let $\{(q_{a_1}, w_{a'_1}), .., (q_{a_t}, w_{a'_t})\}$ be the (q, w) mapping set being found during the current iteration. With $\{(q_{a_1}, w_{a'_1}), .., (q_{a_t}, w_{a'_t})\}$, we set the entries of row$_i$ of $\mathbf{B}$ to 0 for $i \in \{a_1, \ldots, a_t\}$, and set the entries of row$_{i'}$ of $\mathbf{A}''$ to 0 for $i' \in \{a'_1, \ldots, a'_t\}$. We then re-compute the column-sum of $\mathbf{B}$ (resp. $\mathbf{A}''$) for the columns that have not been mapped, and add the computed column-sum to the corresponding vector as its next element. For each distinct vector VB_j of $\mathbf{B}$, if these exists a vector $VA_{j'}$ of $\mathbf{A}''$ that equals VB_j, we can conclude that the plaintext of ed_j is $d_{j'}$. The above procedure is performed in every iteration, until no more new (q, w) mappings are found.

4.3 Description of LEAP

LEAP is shown in Figure 1, where the **Occurrence(**$C, \mathbf{M}, \mathbf{M}', \mathbf{A}'', \mathbf{B}$**)** algorithm, shown in Algorithm 1, serves as a subroutine of the attack.

Step 0 initializes several variables that are used in the following steps. In particular, C_{new} is used to record newly found (ed, d) mappings in **Step 6**, **Step 7** and **Step 8**; R_{new} is used to record newly found (q, w) mappings in **Step 5**. C is used to accumulate (ed, d) mappings, and R is used to accumulate (q, w) mappings. **Step 2** uses **Method 1** as described in Section 4.2. **Step 3** utilizes the $n \times n$ ed-occurrence matrix $\mathbf{M}$ and the $n' \times n'$ d-occurrence matrix $\mathbf{M}'$ to find more (ed, d) mappings based on **Method 2**. **Step 5**, **Step**

Algorithm 1: Occurrence(C,$\mathbf{M}$,$\mathbf{M}'$,$\mathbf{A}''$,$\mathbf{B}$)

Input: A known (ed, d) mapping set C; an $n \times n$ ed-occurrence matrix $\mathbf{M}$ and an $n' \times n'$ d-occurrence matrix $\mathbf{M}'$, where $n' < n$; an $m \times n'$ matrix $\mathbf{A}''$ and an $m \times n$ matrix $\mathbf{B}$.

Output: A set of (ed, d) mappings;

1 Initialize a set $S = \{1\}$ and a set $C' = \emptyset$;
2 Set $C' = C$;
3 **while** $S \neq \emptyset$ **do**
4   Set $S = \emptyset$;
5   **for** *each unmapped* $d_{y_{j'}}$ *for* $j' \in [n']$ **do**
6   Set candidate ED be any unmapped ed_j for $j \in [n]$ satisfying $c'_{j'} = c_j$, where $c'_{j'}$ is the column$_{j'}$-sum of $\mathbf{A}''$ and c_j is the column$_j$-sum of $\mathbf{B}$;
7   **for** *each* ed_j *in* ED **do**
8   **for** *known mappings* $(ed_k, d_{y_{k'}})$ *in* C' **do**
9   **if** $M_{j,k} \neq M'_{j',k'}$ **then**
10   remove ed_j from ED;
11   **end**
12   **end**
13   **end**
14   **if** *only one* ed_j *remains in* ED **then**
15   add $(ed_j, d_{y_{j'}})$ to S;
16   set $C' = C' \cup S$.
17   **end**
18   **end**
19 **end**
20 return S;

6 and **Step 7** are based on **Method 3**, **Method 4**, and **Method 5** respectively.

We then give the description of Algorithm 1. The main idea is to utilize known (ed, d) mappings to find more (ed, d) mappings. It takes known (ed, d) mappings as input. This algorithm is based on **Method 2**, which is built on the following two observations (as described in Section 4.2):

- If ed_k is the encrypted version of $d_{y_{k'}}$ and ed_j is the encrypted version of $d_{y_{j'}}$, then the equation $M_{j,k} = M'_{j',k'}$ holds. However $M_{j,k} = M'_{j',k'}$ does not imply that ed_k is the encrypted version of $d_{y_{k'}}$ and ed_j is the encrypted version of $d_{y_{j'}}$.
- For a known $(ed_k, d_{y_{k'}})$ mapping and a (unmapped) $d_{y_{j'}}$, we can obtain a new mapping $(ed_j, d_{y_{j'}})$ if ed_j is the only candidate satisfying (1) $M_{j,k} = M'_{j',k'}$, and (2) $c'_{j'} = c_j$, where $c'_{j'}$ is the column$_{j'}$-sum of $\mathbf{A}''$ and c_j is the column$_j$-sum of $\mathbf{B}$.

The first observation is utilized in Lines 7-13 in Algorithm 1 to filter some ed_j from the candidate set. The second observation is utilized in Lines 14-16 in Algorithm 1 to obtain the $(ed_j, d_{y_{j'}})$ mapping.

4.4 Analysis of LEAP

Step 2 is the starting point of LEAP. The (ed, d) mappings found in this step serve as initial (ed, d) mappings to bootstrap **Step 3**. **Step 3** aims to find more (ed, d) mappings based on the $n \times n$ ed-occurrence matrix M and the $n' \times n'$ d-occurrence matrix M', which crucially relies on the (ed, d) mappings found in **Step 2**. The more (ed, d) mappings are found in **Step 2**, the more (ed, d) mappings would be found in **Step 3**.

The task of **Step 5** is to find (q, w) mappings. This is the only step that aims to recover (q, w) mappings. The effectiveness of this step depends strongly on the size of the (ed, d) mapping set C accumulated in **Step 2** and **Step 3**, which is used to generate the C-column-mapped submatrix pair $(\mathbf{B}_c, \mathbf{A}''_c)$ from $(\mathbf{B}, \mathbf{A}'')$. If more (ed, d) mappings are accumulated in C, $\mathbf{B}_c$ (resp. $\mathbf{A}''_c$) is wider, and the probability of finding unique rows in it becomes higher. Hence, it is important to find as many (ed, d) mappings as possible before executing **Step 5**.

On the other hand, the (q, w) mappings found in **Step 5** are utilized to find more (ed, d) mappings in **Step 6**. Similarly, a larger size of R leads to higher $\mathbf{B}_r$ (resp. $\mathbf{A}''_r$), and a higher probability of finding unique columns in it.

Step 8 is similar to **Step 3**, which employs $\mathbf{M}$ and $\mathbf{M}'$ to find more (ed, d) mappings. A larger size of C leads to more (ed, d) mappings to be found.

We define *keyword recovery rate* as the percentage of keywords from $\mathbf{W}'$ (where $\mathbf{W}' = \{w_{x_1}, ..., w_{x_{m'}}\}$ is as defined Section 4.1) that have been mapped to the query tokens. In other words, keyword recovery rate is the percentage of rows of $\mathbf{A}''$ that can be uniquely mapped to the rows of $\mathbf{B}$. We further define *accuracy rate of recovered keywords* as the percentage of recovered keywords that are correctly mapped to query tokens, and *correct keyword recovery rate* as the percentage of keywords from $\mathbf{W}'$ that have been *accurately* mapped to the query tokens.

Similarly, we define *document recovery rate* as the percentage of known documents which are mapped to their encrypted versions. Document recovery rate is the percentage of columns of $\mathbf{A}'$ that can be uniquely mapped to the columns of $\mathbf{B}$. We further define *accuracy rate of recovered document* as the percentage of recovered documents that are correctly mapped to encrypted documents, and *correct document recovery rate* as the percentage of known documents that have been accurately mapped to the encrypted documents.

We take the following example to demonstrate the differences of the above definitions. For a known document set with 100 keywords, suppose there exists an attack that returns 80 (q, w) mappings with 40 correct mappings. In this case, the keyword recovery rate is 80%, the accuracy rate of recovered keywords is 50%, and the correct keyword recovery rate is 40%. Intuitively, keyword recovery rate alone cannot reflect the power of an attack, since the result may contain false positives. With the accuracy rate of recovered keywords, one can see how "good" the result is. The correct keyword recovery rate reflects how effective of an attack. The same holds for the case of document recovery.

5 EXPERIMENTS

Here we report the experimental results of LEAP, which is the first attack that targets exact recovery of (q, w) mappings and (ed, d) mappings from partially-known documents and information leakage of EDESE schemes.

Input: An $m' \times n'$ document-keywrod matrix $\mathbf{A}'$ and an $m \times n$ encrypted document-query token matrix $\mathbf{B}$, where $m' < m$ and $n' < n$.
Output: A set of (q, w) mapping and a set of (ed, d) mapping.

- **Step 0** *(Initialization):* Initialize a counter $ct = 1$, four sets $C_{new} = \emptyset$, $R_{new} = \emptyset$, $C = \emptyset$, $R = \emptyset$, and two matrices $\mathbf{B}_{map}$ and $\mathbf{A}''_{map}$.
- **Step 1** *(Extend A'):* Extend the $m' \times n'$ matrix $\mathbf{A}'$ to an $m \times n'$ matrix $\mathbf{A}''$ with the new entries $A''_{i,j} = 0$ for $i \in \{m' + 1, \ldots, m''\}$ and $j \in [n']$. Set $\mathbf{B}_{map} = \mathbf{B}$ and $\mathbf{A}''_{map} = \mathbf{A}''$.
- **Step 2** *(Find (ed, d) mappings)*: For each $j \in [n]$, do: (1) Initialize a vector VB_j for column$_j$ of $\mathbf{B}$; (2) Compute column$_j$-sum as c_j, and set $VB_j[1] = c_j$. Similarly, for each $j' \in [n']$, do: (1) Initialize a vector $VA_{j'}$ for column$_{j'}$ of $\mathbf{A}''$; (2) Compute the column$_{j'}$-sum as $c'_{j'}$, set $VA_{j'}[1] = c'_{j'}$. For each VB_j that is unique among $\{VB_j\}_{j \in [n]}$, if there exists a $VA_{j'}$ such that $VB_j = VA_{j'}$ (where $j' \in [n']$), add $(ed_j, d_{y_{j'}})$ into C.
- **Step 3** *(Find more (ed, d) mappings)*: Compute the $n \times n$ ed-occurrence matrix $\mathbf{M}$ and the $n' \times n'$ d-occurrence matrix $\mathbf{M}'$, run **Occurrence(C, $\mathbf{M}$, $\mathbf{M}'$, $\mathbf{A}''_{map}$, $\mathbf{B}_{map}$)** to obtain a (ed, d) mapping set S. Add S into C.
- **Step 4**: Set $ct = ct + 1$, and $R_{new} = C_{new} = \emptyset$.
- **Step 5** *(Find (q, w) mappings)*: Generate C-column-mapped submatrix pair $(\mathbf{B}_c, \mathbf{A}''_c)$ from $(\mathbf{B}_{map}, \mathbf{A}''_{map})$. For row$_i$ of $\mathbf{B}_c$ that has unique bit-string among all the rows, find row$_{i'}$ of $\mathbf{A}''_c$ that has the same bit-string as the row$_i$ of $\mathbf{B}_c$. If found, add $(q_i, w_{x_{i'}})$ into R_{new} and R respectively;
- **Step 6** *(Find more (ed, d) mappings)*: Generate R-row-mapped submatrix pair $(\mathbf{B}_r, \mathbf{A}''_r)$ from $(\mathbf{B}_{map}, \mathbf{A}''_{map})$. For column$_j$ of $\mathbf{B}_r$ that has unique bit-string among all the columns, find column$_{j'}$ of $\mathbf{A}''_r$ that has the same bit-string as the column$_j$ in $\mathbf{B}_r$. If found, add $(ed_j, d_{y_{j'}})$ into C_{new} and C respectively;
- **Step 7** *(Find more (ed, d) mappings)*: Set the entries of all the matched rows in $\mathbf{B}$ and $\mathbf{A}''$ to 0. For each column$_j$ of $\mathbf{B}$ that hasn't been mapped, (re-)compute its column$_j$-sum as c_j, and set $VB_j[ct] = c_j$. Similarly, for each column$_{j'}$ of $\mathbf{A}''$ that hasn't been mapped, compute its column$_{j'}$-sum as $c'_{j'}$, set $VA_{j'}[ct] = c_{j'}$. For each VB_j that is unique among $\{VB_j\}_{j \in S_{up}}$, if there exists a $VA_{j'}$ such that $VB_j = VA_{j'}$ (where $j' \in S'_{up}$), add $(ed_j, d_{y_{j'}})$ into C_{new} and C respectively, where S_{up}, S'_{up} are the index sets of the unmapped columns in $\mathbf{B}$ and $\mathbf{A}''$ respectively;
- **Step 8** *(Find more (ed, d) mappings)*: Run **Occurrence(C, $\mathbf{M}$, $\mathbf{M}'$, $\mathbf{A}''_{map}$, $\mathbf{B}_{map}$)** to obtain a (ed, d) mapping set S'. Add S' into C_{new} and C respectively.
- **Step 9**: If ($R_{new} \neq \emptyset$ *or* $C_{new} \neq \emptyset$), execute **Step 4**; otherwise, execute **Step 10**.
- **Step 10**: Output R as the set of recovered (q, w) mappings and C as the set of recovered (ed, d) mappings.

Figure 1: Description of LEAP

5.1 Setting

In our experiments, we use the Enron email database [13] as in previous studies [5, 39, 44]. The database consists of 30,109 emails of 150 employees from the Enron corporation, which were sent between 2000-2002. We treat each email as a single document. The document universe consists of all 30,109 emails. We adopt the same method as in [5, 44] to extract keywords from the emails. In particular, all words are first processed according to the standard Porter stemming algorithm [38] so as to generate a keyword set, and the stop words (such as "a", "the", "to", etc.) are removed from the keyword set. Similar to [5, 44], in our experiments, we chose the top 5,000 most frequent keywords from the keyword set as the keyword universe for each case in our experiments. For the case where the number of keywords of the leaked documents is less than 5,000, we choose all the keywords as the keyword universe corresponding to the known documents. The documents known to the attacker are chosen uniformly from the document universe (i.e., 30,109 emails in the Enron email dataset), and the percentage of leaked documents varies from 100% to 0.1%.

5.2 Keyword Recovery

In our experiments for keyword recovery, we compare with the PW-U attack, the PW-P attack, the CGPR attack and the attack in [2] (BKM). The PW-U attack and the PW-P attack target EDESE schemes and are based on L2 leakage. These are the most relevant attacks to LEAP. As noted in [39], the PW-U attack and the PW-P attack can also work without the prior knowledge of the dataset [3]. However, they need an extra dataset as the training set. In order to make the attacks effective, the extra dataset needs to share similar property as the target dataset. The CGPR attack is another attack that recovers query token with partial known document set. In contrast, it mainly utilizes L1 leakage, rather than L2 leakage. The BKM attack is the state-of-the-art in keyword recovery

[3] Indeed, the attacks can tolerate imperfect auxiliary information.

TABLE II: Comparison with the CGPR attack [5], the attacks in [39] and the BKM attack [2] [1]

Dataset Knowledge		No. of Recovered Keywords (Keyword Recovery Rate)					No. of Correctly Recovered Keywords (Accuracy Rate of Recovered Keywords) (Correct Keyword Recovery Rate)				
N-LD (P-LD)	No. of KW	CGPR	PW		BKM	LEAP	CGPR	PW		BKM	LEAP
			U	P				U	P		
30 (0.1%)	1,144	1 (0.09%)	1,144 (100%)	1,144 (100%)	0 (0%)	132 (11.54%)	0 (0%) (0%)	5 (0.44%) (0.44%)	2 (0.17%) (0.17%)	0 (0%) (0%)	132 (100%) (11.54%)
150 (0.5%)	2,315	5 (0.22%)	2,315 (100%)	2,315 (100%)	1 (0.04%)	860 (37.15%)	1 (20%) (0.04%)	7 (0.3%) (0.3%)	21 (0.91%) (0.91%)	1 (100%) (0.04%)	860 (100%) (37.15%)
301 (1%)	3,318	4 (0.12%)	3,318 (100%)	3,318 (100%)	2 (0.06%)	1,754 (52.86%)	2 (50%) (0.06%)	15 (0.45%) (0.45%)	51 (1.54%) (1.54%)	2 (100%) (0.06%)	1,754 (100%) (52.86%)
1505 (5%)	4,889	10 (0.2%)	4,889 (100%)	4,889 (100%)	56 (1.15%)	4,540 (92.86%)	5 (50%) (0.1%)	27 (0.55%) (0.55%)	621 (12.7%) (12.7%)	53 (94.64%) (1.08%)	4,540 (100%) (92.86%)
3,010 (10%)	4,991	21 (0.42%)	4,991 (100%)	4,991 (100%)	699 (14.01%)	4,904 (98.26%)	7 (33.33%) (1.4%)	38 (0.76%) (0.76%)	1,638 (32.82%) (32.82%)	671 (95.99%) (13.44%)	4,904 (100%) (98.26%)
6,021 (20%)	5,000	42 (0.84%)	5,000 (100%)	5,000 (100%)	3,532 (70.64%)	4,957 (99.14%)	7 (16.67%) (0.14%)	82 (1.64%) (1.64%)	2,322 (46.44%) (46.44%)	3,513 (99.46%) (70.26%)	4,957 (100%) (99.14%)
9,032 (30%)	5,000	68 (1.36%)	5,000 (100%)	5,000 (100%)	4,468 (89.36%)	4,961 (99.22%)	7 (10.29%) (0.14%)	93 (1.86%) (1.86%)	2,441 (48.82%) (48.82%)	4,458 (99.78%) (89.16%)	4,961 (100%) (99.22%)
12,043 (40%)	5,000	83 (1.66%)	5,000 (100%)	5,000 (100%)	4,527 (90.54%)	4,965 (99.30%)	8 (9.64%) (0.16%)	148 (2.96%) (2.96%)	2,469 (49.38%) (49.38%)	4,520 (99.85%) (90.4%)	4,965 (100%) (99.3%)
15,054 (50%)	5,000	97 (1.94%)	5,000 (100%)	5,000 (100%)	4,592 (91.84%)	4,966 (99.32%)	8 (8.25%) (0.16%)	219 (4.38%) (4.38%)	2,948 (58.96%) (58.96%)	4,587 (99.89%) (91.74%)	4,966 (100%) (99.32%)
30,109 (100%)	5,000	4,611 (92.22%)	5,000 (100%)	5,000 (100%)	4,916 (98.32%)	4,973 (99.46%)	4,610 (99.98%) (92.2%)	4,979 (99.58%) (99.58%)	3,169 (63.38%) (63.38%)	4,916 (100%) (98.32%)	4,973 (100%) (99.46%)

[1] "L1" indicates that the attack is based on L1 leakage, "L2" indicates that the attack is based on L2 leakage. "N-LD" denotes the number of leaked documents of the dataset, "P-LD" denotes the percentage of leaked documents of the dataset. "No. of KW " denotes the top 5,000 most frequent keywords corresponding to the leaked documents (if the total number of keywords is less than 5,000, we take all the keywords as the keyword universe, as in the cases where only 10%, 5%, 1%, 0.5% and 0.1% of documents are leaked). "No. of Recovered Keywords" denotes the number of recovered keywords, "Keyword Recovery Rate" denotes the percentage of keywords that are mapped to query tokens. "No. of Correctly Recovered Keywords" denotes the number of correctly recovered keywords out of the recovered keywords, "Accuracy Rate of Recovered Keywords" denotes the percentage of correctly recovered keywords from the recovered keywords. "Correct Keyword Recovery Rate" denotes the percentage of correct recovered keywords out of the keyword universe corresponding to the known documents. "U" stands for the PW-U attack which is based on the Umeyama's algorithm [39], "P" stands for the PW-P attack which is based on the PATH algorithm [39].

attack that assumes exact knowledge of a subset of the documents. Here, we compare with the CGPR attack and the BKM attack for completeness.

Table II shows the comparison among the CGPR attack, the PW-U attack, the PW-P attack, the BKM attack and LEAP in terms of recovered keywords. We evaluate the number (resp. percentage) of recovered keywords, the number of correctly recovered keywords, the accuracy rate of recovered keywords, and the correct keyword recovery rate by varying the percentage of documents known to the attacker from 100% to 0.1%. As noted in Section 4.4, the keyword recovery rate reflects how many keywords from the keyword universe corresponding to the known documents could be recovered. The accuracy rate of recovered keywords reflects how many recovered keywords are correct. The correct keyword recovery rate reflects how many keywords can be recovered correctly.

The PW-U attack recovers all the keywords in the keyword universe corresponding to the leaked documents no matter how many documents are leaked. In terms of accuracy rate of recovered keywords, 99.58% of the (q, w) mappings are correctly recovered by the PW-U attack when given the entire database. However, the accuracy rate of recovered keywords of the PW-U attack drops dramatically when the percentage of leaked documents are less than 50%. As shown in Table II, only 4.38% of the recovered (q, w) mappings are correct with the PW-U attack even given 50% of the dataset. When given 10% of the dataset, only 0.76% of the (q, w) mappings recovered by the PW-U attack are correct. This shows that though the keyword recovery rate of the PW-U attack is 100%, however, most of the recovered (q, w) mappings are not correct when only partial knowledge of the dataset is available to the attacker.

The PW-P attack, similar to the PW-U attack, recovers 100% the keywords when the percentage of the leaked documents varies from 100% to 0.1%. For 100% leaked documents, the accuracy rate of recovered keywords is 63.38%, which is less than that of the PW-U attack. When given partial knowledge of the dataset, however, the PW-P attack performs better than the PW-U attack in terms of accuracy rate of recovered keywords. Given 50% of the documents, the accuracy rate of recovered keywords of the PW-P attack is 58.96%, compared to 4.38% of the PW-U attack. For the case where 10% of the documents are leaked, the accuracy rate of recovered keywords of the PW-P attack is 32.82%, compared to 0.76% of the PW-U attack.

In terms of keyword recovery rate, the CGPR attack recovers 92.22% of the keywords when given the entire dataset; however, the rate drops dramatically as the fraction of leaked documents decreases. In particular, 1.94% of the keywords are recovered by the CGPR attack when given 50% of the documents. The CGPR attack can only recover 0.42% of the keywords given 10% of the dataset. In terms of correctly recovered keywords, when give the entire dataset, 99.98% of the (q, w) mappings recovered by the CGPR attack are correct. In other words, 92.2% of the keywords can be accurately recovered. When the percentage of leaked documents are less than 50%, however, only very few keywords are correctly recovered by the CGPR attack. This indicates that the CGPR attack does not perform well when the attacker has only partial knowledge of the document set.

For the BKM attack, in terms of keyword recovery rate, it recovers 98.32% of the keywords when given 100% dataset. The keyword recovery rate drops dramatically when the percentage of leaked documents are less than 20%. When given 10% of the dataset, 14.01% of the keywords are recovered by the BKM attack. It only recovers 0.06% of the keywords given 1% of the dataset. In terms of correctly recovered keywords, all of the (q, w) mappings recovered by the BKM attack are correct when give the entire dataset. The number of correctly recovered keywords drops dramatically when the percentage of leaked documents are less than 10%. This shows that the BKM attack does not perform well when given partial knowledge of the document set.

The keyword recovery rate of LEAP varies from 99.46% to 11.54% with the percentage of leaked documents varying from 100% to 0.1%. For each case that we test, LEAP achieves 100% accuracy rate of recovered keywords, indicating that every recovered (q, w) mapping is correct. In particular, given 10% of the dataset, the number of correctly recovered keywords of LEAP is 4,904, as compared to 1,638 of the PW-P attack and 38 of the PW-U attack. When given 0.1% of the dataset, LEAP correctly recovers 132 (q, w) mappings, as compared to 2 of the PW-P attack and 5 of the PW-U attack. This demonstrates that LEAP is significantly more powerful than the PW-P attack and the PW-U attack in terms of correctly recovered keywords. Though the PW-P attack and the PW-U attack could recover 100% the keywords, most of the recovered (q, w) mappings of the PW-U attack and less than half of the recovered (q, w) mappings of the PW-P attack are wrong. In this sense, LEAP is the most powerful among these attacks.

5.3 Document Recovery

Since LEAP achieves keyword recovery and document recovery simultaneously, we record the experimental results of document recovery when carrying out our experiments, which are shown in Table III. We record the number (resp. percentage) of recovered documents, the number of correctly recovered documents, the accuracy rate of recovered documents, the correct document recovery rate, the number of recovered documents using only recovered keywords, by varying the percentage of documents known to the attacker from 100% to 0.1%. Table III shows that LEAP recovers most of the encrypted documents for each case. In addition, LEAP achieves 100% accuracy rate of recovered documents in the sense that all of the recovered documents are correct. Specifically, when given only 10% of the entire dataset, 92.16% of the 3,010 encrypted documents can be accurately recovered by LEAP. Given only 0.5% of the dataset, LEAP still recovers 91.33% of the 150 encrypted documents. This demonstrates that LEAP is very powerful in recovering the mapping between (plaintext) documents and the encrypted documents.

We also compare the number of recovered documents using our document recovery method with that using only recovered keywords. As shown in Table III, for the cases where 5%, 1%, 0.5% and 0.1% of the dataset is leaked, our document recovery method recovers more documents than that using only recovered keywords. This demonstrates that our document recovery method works better than purely using recovered keywords when the number of leaked documents is small.

5.4 Property of Uniqueness

LEAP crucially relies on the uniqueness of the columns and rows in $\mathbf{B}$ and $\mathbf{A}''$ in finding unique row mappings and unique column mappings between $\mathbf{B}$ and $\mathbf{A}''$. Such uniqueness property of the columns and rows in $\mathbf{B}$ and $\mathbf{A}''$ mainly relies on the following three factors: (1) column-sum, (2) d-occurrence matrix and ed-occurrence matrix, and (3) bit-string of column and bit-string of row. In particular, the number of (ed, d) pairs found in **Step 2** determines the effectiveness of LEAP. This is so because the (ed, d) pairs found in **Step 2** serve as an input to subsequent steps. The more such (ed, d) pairs exist, the more (q, w) mappings and (ed, d) mappings can be recovered in subsequent steps.

To see how the number of leaked documents affect the uniqueness of the columns and rows in $\mathbf{B}$ and $\mathbf{A}''$, we record the number of unique columns found in **Step 2** (denoted as *initial unique column*), the number of unique columns found once **Step 9** finished (denoted as *final unique column*), and the number of unique rows found in **Step 5** (denoted as *unique row*), respectively. Table IV shows the numbers of the initial unique columns, final unique columns and unique rows, respectively, found by LEAP. It demonstrates that the more leaked documents, the more initial unique columns, final unique columns and unique rows can be found. The initial unique columns are the starting point of LEAP, only 11 initial unique columns are found for 10% leaked documents and 1 for 1% leaked documents; nevertheless they are enough for "bootstrapping" the subsequent steps.

TABLE III: Document Recovery [1]

No. of Leaked Doc. /(Per. of Leaked Doc.)	No. of Recovered Documents /Document Recovery Rate	No. of Correctly Recovered Documents /Accuracy Rate of Recovered Documents /Correct Document Recovery Rate	No. of Recovered Documents using RK
30 / (0.1%)	29 / 96.66%	29 / 100% / 96.66%	28
150 / (0.5%)	137 / 91.33%	137 / 100% / 91.33%	134
301 / (1%)	273 / 90.69%	273 / 100% / 90.69%	269
1,505 / (5%)	1,394 / 92.62%	1,394 / 100% / 92.62%	1,392
3,010 / (10%)	2,774 / 92.16%	2,774 / 100% / 92.16%	2,774
6,021 / (20%)	5,548 / 92.14%	5,548 / 100% / 92.14%	5,548
9,032 / (30%)	8,340 / 92.34%	8,340 / 100% / 92.34%	8,340
12,043 / (40%)	11,132 / 92.43%	11,132 / 100% / 92.43%	11,132
15,054 / (50%)	13,915 / 92.43%	13,915 / 100% / 92.43%	13,915
30,109 / (100%)	27,808 / 92.35%	27,808 / 100% / 92.35%	27,808

[1] "No. of Leaked Doc." and "Per. of Leaked Doc." hold the same meaning as that in Table II. "No. of Recovered Documents" denotes the number of recovered documents, "Document Recovery Rate" denotes the percentage of encrypted documents that are recovered. "No. of Correctly Recovered Documents" denotes the number of correctly recovered encrypted documents out of the encrypted documents, "Accuracy Rate of Recovered Documents" denotes the percentage of correctly recovered documents from the recovered documents. "Correct Document Recovery Rate" denotes the percentage of correct recovered documents out of the encrypted document universe. "No. of Recovered Documents using RK" denotes the number of recovered documents purely using the knowledge of recovered keywords (i.e., not applying our document recovery method).

TABLE IV: Uniqueness of the columns and rows in matrices B and A″ [1]

Dataset Knowledge		No. of Initial Unique Col. /(Per. of Initial Unique Col.)	No. of Final Unique Col. /(Per. of Final Unique Col.)	No. of Unique Rows /(Per. of Unique Rows)
No. of Leaked Doc. /(Per. of Leaked Doc.)	No. of KW			
30 / (0.1%)	1,144	1 / 3.33%	29 / 96.66%	132 / 11.54%
150 / (0.5%)	2,315	1 / 0.66%	137 / 91.33%	860 / 37.15%
301 / (1%)	3,318	1 / 0.33%	273 / 90.69%	1,754 / 52.86%
1,505 / (5%)	4,889	8 / 0.53%	1,394 / 92.62%	4,540 / 92.86%
3,010 / (10%)	4,991	11 / 0.36%	2,774 / 92.16%	4,904 / 98.26%
6,021 / (20%)	5,000	17 / 0.28%	5,548 / 92.14%	4,957 / 99.14%
9,032 / (30%)	5,000	27 / 0.29%	8,340 / 92.34%	4,961 / 99.22%
12,043 / (40%)	5,000	33 / 0.27%	11,132 / 92.43%	4,965 / 99.3%
15,054 / (50%)	5,000	38 / 0.25%	13,915 / 92.43%	4,966 / 99.32%
30,109 / (100%)	5,000	73 / 0.24%	27,808 / 92.35%	4,973 / 99.46%

[1] "No. of Leaked Doc.", "Per. of Leaked Doc." and "No. of KW" hold the same meaning as that in Table II. " No. pf Initial Unique Col." denotes the number of columns that are found as initial unique column, "Per. of Initial Unique Col." denotes the percentage of columns that are found as initial unique columns. "No. of Final Unique Col." denotes the number of columns that are found as final unique columns, "Per. of Final Unique Col." denotes the percentage of columns that are found as final unique columns. "No. of Unique Rows" denotes the number of unique rows, "Per. of Unique Rows" denotes the percentage of rows that are found as unique rows.

5.5 Scalability

LEAP mainly relies on matrix operations. To deal with larger matrix as the document set size increases, we mainly utilize the following two approaches. First, we cache the intermediate results when preparing **B**, **A″** and **M**, which could be reused for each case of our experiments. During the preparation, the following procedures can be parallelized: (1) the extraction for the relationship between encrypted documents and query tokens from L2 leakage; (2) the relationship between documents and keywords from leaked documents. Second, we divide one matrix into submatrices during the execution of LEAP. In particular, before starting **Step 2**, we divide **B** and **A″** into a set of submatrices while keeping the number of rows unchanged. These submatrices can be parallelized during **Step 2**. Similar divide-and-parallelize idea for matrix operations can be applied in subsequent steps.

6 COUNTERMEASURES

In this section, we discuss possible countermeasures against our attack. LEAP crucially relies on **Step 2** and **Step 3** in Figure 1, which is the starting point of this attack. In these steps, the initial tuples of known (ed, d) mappings are prepared so as to bootstrap the subsequent steps (i.e., from **Step 5** to **Step 8**) in Figure 1. The (ed, d) pairs found in **Step 2** serve as an input (i.e., C) to the subroutine **Occurrence(**C**,M,M′, A″,B)** in **Step 3**, which aims to find more (ed, d) mappings. The (ed, d) pairs found in **Step 2** are derived from $(VB_j, VA_{j'})$ pairs where VB_j is unique among its peers and $VB_j = VA_{j'}$. The more such $(VB_j, VA_{j'})$ pairs exist, the more (q, w) mappings and (ed, d) mappings can be recovered. On the other hand, if no such $(VB_j, VA_{j'})$ pair exists, LEAP would fail to find any (q, w) or (ed, d) mappings. Hence, the number of such $(VB_j, VA_{j'})$ pairs found in **Step 2** determines the effectiveness of LEAP. This means an effective countermeasure against LEAP would be to reduce or even eliminate the existence of such $(VB_j, VA_{j'})$ pairs. One possible solution is to add keywords from **W** to existing documents that each query token is attached to more encrypted documents than it should be (i.e., it turns some entries of **B** from 0 to 1). These extra dummy encrypted documents can be filtered out by the user after data decryption. This method is similar to the *padding* solution in [5]. If the dummy documents are added to the point that there exists no unique VB_j, then LEAP would fail. This is because there is no initial (ed, d) mappings to bootstrap the subsequent steps of LEAP.

However, one can use a modified attack similar to the *generalized count attack* described in [5] to partially alleviate the above padding countermeasure. Specifically, we can adopt the modifications as introduced in [5]: (1) for the $n \times n$ ed-occurrence matrix $\mathbf{M}$ and the $n' \times n'$ d-occurrence matrix $\mathbf{M'}$, we modify Line 9 of Algorithm 1 by letting $M[j,k]$ not equal to $M'[j',k']$ but within a window as large as the maximum number of false co-occurrences; (2) We can make an initial guess for the $(VB_j, VA_{j'})$ pair in **Step 2** to start Algorithm 1. If later the algorithm detects an inconsistency, we can guess another $(VB_j, VA_{j'})$ pair. As a result, there is no guarantee that we can achieve keyword recovery and document recovery accurately. In other words, the recovered results may result in false positives.

7 RELATED WORK

The first practical searchable encryption scheme was introduced by Song, Wagner and Perrig [40]. Subsequently, many variants were proposed to improve on performances, security and functionalities [1, 4, 6–12, 15, 16, 22–25, 29, 30, 32, 33, 37, 41]. Most, if not all, SE schemes are designed based on the assumption that certain leakage of information (e.g. L1 leakage, L2 leakage) is acceptable as a trade-off for high efficiency as required for practical usage. An overview of searchable encryption schemes is given in [3].

Various leakage-based attacks have been discovered recently that successfully compromise some of the existing SE schemes [2, 5, 21, 35, 39]. Islam *et al.* [21] demonstrated how access pattern can be used to recover the underlying keywords and documents in SE assuming that an attacker knows either all plaintext documents or keyword distribution. Cash *et al.* [5] categorised SE leakages into different levels, and improved Islam *et al.*'s attack by presenting a more effective leakage-based attack that could work with less knowledge about the user's documents. An active attack that induces a user to insert chosen documents was also introduced in [5]. Pouliot and Wright [39] later proposed new inference attacks on EDESE schemes that demonstrate the consequence of the information leakage of EDESE schemes. Zhang *et al.* [44] later presented a file-injection attack, in which an attacker selectively injects certain documents so as to recover underlying keywords and documents. Our work is closely related to and improves on the inference attacks proposed by Pouliot and Wright [39] in the sense that our attack is the first attack (as far as we know) achieving accurate keyword recovery and document recovery with only partial knowledge of a user's documents and L2 leakage. The passive attack proposed in [35] is also closely related to ours, however, it does not work under the setting we consider in this paper. Recently, Blackstone *et al.* [2] revisited the attacks in [5, 21] and proposed new leakage-abuse attacks. They assumed that the attacker knows the universe of keywords from which the queries are drawn. This is different from our assumption in this paper where we do not require the knowledge of keyword universe.

Kellaris *et al.* [26] stated that access pattern leakage is unavoidable and introduced an attack on keyword recovery based on range queries. Following from Kellaris *et al.*'s attacks, there have also been recent works focusing on reconstruction attacks on range queries [17–19, 28, 31, 34] and k-NN queries [27, 28]. Lacharité *et al.* [31] proposed new attacks with the assumption that the database is dense, while subsequent attacks proposed by Grubbs *et al.* [17] make no such assumption. However, these attacks assume that the queries are either uniformly distributed (as in the Kellaris *et al.*'s attacks), or that the query and approximation of the data distributions are known. Gui *et al.* [19] proposed attacks based on Kellaris *et al.*'s work, but require fewer queries and do not assume uniformly distributed queries. Nevertheless, there are other assumptions such as queries for all possible volume must be observed at least once. Independently, Kornaropoulos *et al.* [27] proposed reconstruction attacks for k-nearest neighbor (k-NN) queries, which are widely used in spatial data databases. The proposed attacks also assume uniformly distributed queries. More recently, Kornaropoulous *et al.* [28] propose attacks that work against both k-NN queries and range queries, and is agnostic to query distribution. The attacks leverage on both the search and access pattern leakages, as opposed to previous attacks that leverage on access pattern leakage only. Poddar *et al.* [36] proposed a new reconstruction attack that utilizes common characteristics in practical applications, that is, file injection and automatic query replay, in conjuction with volume leakage. This means the attack assumes an adversary is able to inject files and replay a query. The attack was tested on Gmail. Recently, Falzon *et al.* [14] explored the threat in two dimensions databases that support range queries and presented a full database reconstruction attack.

8 CONCLUSIONS

In this work, we proposed a new leakage-abuse attack on EDESE schemes termed LEAP. Through LEAP we demonstrated that the underlying keywords of query tokens can be recovered accurately, even with partial knowledge of the document set. Rigorous experiments illustrate that LEAP achieves high correct keyword recovery rate and correct document recovery rate, as compared to the PW-U attack and the PW-P attack. Our findings show that even if a small portion of a document set is known to an attacker, the information leakage (e. g., L2 leakage) of EDESE schemes can be very damaging.

ACKNOWLEDGMENTS

We thank anonymous reviewers for helpful comments. Research supported in part by AXA Research Fund, the National Natural Science Foundation of China (Grant Nos. 62032005, 61972094), the Science Foundation of Fujian Provincial Science and Technology Agency (2020J02016), and the young talent promotion project of Fujian Science and Technology Association. Jian Weng was supported by Major Program of Guangdong Basic and Applied Research Project (Grant No. 2019B030302008), National Key Research and Development Plan of China (Grant No. 2020YFB1005600), National Natural Science Foundation of China (Grant Nos. 61825203, U1736203 and 61732021), and Guangdong Provincial Science and Technology Project (Grant No. 2017B010111005). Yingjiu Li was supported in part by the Ripple University Blockchain Research Initiative.

REFERENCES

[1] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. 2016. Searchable Symmetric Encryption: Optimal Locality in Linear Space via Two-Dimensional Balanced

Allocations. *IACR Cryptology ePrint Archive (and STOC 2016)* 2016/251 (2016), 1–34.

[2] Laura Blackstone, Seny Kamara, and Tarik Moataz. 2020. Revisiting Leakage Abuse Attacks. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020*. The Internet Society.

[3] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. 2014. A Survey of Provably Secure Searchable Encryption. *ACM Comput. Surv.* 47, 2, Article 18 (2014), 51 pages.

[4] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. 2017. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1465–1482.

[5] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-Abuse Attacks Against Searchable Encryption. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM, 668–679.

[6] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very Large Databases: Data Structures and Implementation. In *NDSS 2014*, Vol. 2014. Internet Society.

[7] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *CRYPTO 2013 (LNCS, Vol. 8042)*, Ran Canetti and Juan A. Garay (Eds.). Springer, 353–373.

[8] David Cash and Stefano Tessaro. 2014. The Locality of Searchable Symmetric Encryption. In *EUROCRYPT 2014 (LNCS, Vol. 8441)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.). Springer, 351–368.

[9] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 1038–1055.

[10] Yan-Cheng Chang and Michael Mitzenmacher. 2005. Privacy Preserving Keyword Searches on Remote Encrypted Data. In *ACNS 2005 (LNCS, Vol. 3531)*, John Ioannidis, Angelos D. Keromytis, and Moti Yung (Eds.). Springer, 442–455.

[11] Melissa Chase and Seny Kamara. 2010. Structured Encryption and Controlled Disclosure. In *ASIACRYPT 2010 (LNCS, Vol. 6477)*, Masayuki Abe (Ed.). Springer, 577–594.

[12] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *ACM CCS 2006*, Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati (Eds.). ACM, 79–88.

[13] Enron email dataset. 2019. https://www.cs.cmu.edu/~./enron/. Accessed: 2019-11-23.

[14] Francesca Falzon, Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia. 2020. Full Database Reconstruction in Two Dimensions. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. 443–460. https://doi.org/10.1145/3372297.3417275

[15] Ben A. Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M. Bellovin. 2015. Malicious-Client Security in Blind Seer: A Scalable Private DBMS. In *IEEE S & P 2015*. IEEE Computer Society, 395–410.

[16] Eu-Jin Goh. 2003. Secure Indexes. IACR Cryptology ePrint Archive, Report 2003/216. http://eprint.iacr.org/2003/216/.

[17] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2018. Pump up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 315–331.

[18] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2019. Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks. In *2019 IEEE Symposium on Security and Privacy, S&P 2019*. IEEE, 1067–1083.

[19] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. 2019. Encrypted Databases: New Volume Attacks against Range Queries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 361–378. https://doi.org/10.1145/3319535.3363210

[20] Warren He, Devdatta Akhawe, Sumeet Jain, Elaine Shi, and Dawn Xiaodong Song. 2014. ShadowCrypt: Encrypted Web Applications for Everyone. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. 1028–1039.

[21] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *NDSS 2012*. The Internet Society.

[22] Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2013. Outsourced symmetric private information retrieval. In *ACM CCS'13*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM, 875–888.

[23] Seny Kamara and Tarik Moataz. 2019. Computationally Volume-Hiding Structured Encryption. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11477)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, 183–213.

[24] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. 2018. Structured Encryption and Leakage Suppression. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10991)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, 339–370.

[25] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *ACM CCS'12*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM, 965–976.

[26] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 1329–1340.

[27] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2019. Data Recovery on Encrypted Databases with k-Nearest Neighbor Query Leakage. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 1033–1050. https://doi.org/10.1109/SP.2019.00015

[28] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2020. The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 1223–1240. https://doi.org/10.1109/SP40000.2020.00029

[29] Kaoru Kurosawa. 2014. Garbled Searchable Symmetric Encryption. In *FC 2014 (LNCS, Vol. 8437)*, Nicolas Christin and Reihaneh Safavi-Naini (Eds.). Springer, 234–251.

[30] Kaoru Kurosawa and Yasuhiro Ohtaki. 2012. UC-Secure Searchable Symmetric Encryption. In *FC'12 (LNCS, Vol. 7397)*, Angelos D. Keromytis (Ed.). Springer, 285–298.

[31] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2018. Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage. In *2018 IEEE Symposium on Security and Privacy, S&P 2018, Proceedings*. IEEE Computer Society, 297–314.

[32] Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shifeng Sun, Dongxi Liu, and Cong Zuo. 2018. Result Pattern Hiding Searchable Encryption for Conjunctive Queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 745–762.

[33] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. 2014. Dynamic Searchable Encryption via Blind Storage. In *IEEE S&P 2014*. IEEE Computer Society, 639–654.

[34] Jianting Ning, Geong Sen Poh, Xinyi Huang, Robert Deng, Shuwei Cao, and Ee-Chien Chang. 2020. Update recovery attacks on encrypted database within two updates using range queries leakage. *IEEE Transactions on Dependable and Secure Computing, DOI: 10.1109/TDSC.2020.3015997* (2020).

[35] Jianting Ning, Jia Xu, Kaitai Liang, Fan Zhang, and Ee-Chien Chang. 2019. Passive Attacks Against Searchable Encryption. *IEEE Transactions Trans. Information Forensics and Security* 14, 3 (2019), 789–802.

[36] Rishabh Poddar, Stephanie Wang, Jianan Lu, and Raluca Ada Popa. 2020. Practical Volume-Based Attacks on Encrypted Databases. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 354–369. https://doi.org/10.1109/EuroSP48549.2020.00030

[37] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *SOSP 2011*, Ted Wobber and Peter Druschel (Eds.). ACM, 85–100.

[38] Martin F. Porter. 2006. An algorithm for suffix stripping. *Program* 40, 3 (2006), 211–218. https://doi.org/10.1108/00330330610681286

[39] David Pouliot and Charles V. Wright. 2016. The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 1341–1352. https://doi.org/10.1145/2976749.2978401

[40] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. 2000. Practical Techniques for Searches on Encrypted Data. In *IEEE S&P '00*. IEEE Computer Society, 44.

[41] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. 2014. Practical Dynamic Searchable Encryption with Small Leakage. In *NDSS 2014*. The Internet Society. http://www.internetsociety.org/events/ndss-symposium-2014

[42] Shinji Umeyama. 1988. An Eigendecomposition Approach to Weighted Graph Matching Problems. *IEEE Trans. Pattern Anal. Mach. Intell.* 10, 5 (1988), 695–703. https://doi.org/10.1109/34.6778

[43] Mikhail Zaslavskiy, Francis R. Bach, and Jean-Philippe Vert. 2009. A Path Following Algorithm for the Graph Matching Problem. *IEEE Trans. Pattern Anal. Mach. Intell.* 31, 12 (2009), 2227–2242. https://doi.org/10.1109/TPAMI.2008.245

[44] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2016. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 707–720. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/zhang