Passive Attacks Against Searchable Encryption

Jianting Ning, Jia Xu, Kaitai Liang, *Member, IEEE*, Fan Zhang, *Member, IEEE*, and Ee-Chien Chang

***Abstract*— Searchable encryption (SE) provides a privacy-preserving mechanism for data users to search over encrypted data stored on a remote server. Researchers have designed a number of SE schemes with high efficiency yet allowing some degree of leakage profile to the remote server. The leakage, however, should be further measured to allow us to understand what types of attacks an SE scheme would encounter. This paper considers passive attacks that make inferences based on prior knowledge and observations on queries issued by users. This is in contrast to previously studied active attacks that adaptively inject files and queries. We consider several assumptions on the types or prior knowledge the attacker possessed and propose a few passive attacks. In particular, under the "full-fledged" assumption, the keyword recovery rate of our attack is optimal in the sense that it is equal to the theoretical upper bound. We further present several enhanced attacks under other weaker assumptions on various levels of the prior knowledge that the attacker can obtain, in which the keyword recovery rates are optimal or nearly optimal (i.e., approaching the theoretical upper bound). In addition, we provide extensive experiments to show the "power" of our passive attacks. This paper highlights the importance of minimizing the prior knowledge of a server and the leakage of search queries. It also shows that simply distorting the frequency of the keyword to hold against our passive attacks may not scale well.**

***Index Terms*— Searchable symmetric encryption, passive attacks, search query privacy, leakage of file-access pattern.**

I. Introduction

TO DATE the advanced cloud-based technologies provide flexible data outsourcing service for the Internet users. The service, however, may incur the concern on the leakage of personal sensitive data, due to the fact that the data is out of data owner's premises. To guarantee the confidentiality of the data, one of the choices is using encryption techniques. Although protecting personal data from being read by malicious cloud server, traditional encryption mechanisms limit further operations over encrypted data. In particular, if all data are encrypted and remotely stored in a cloud server, how can the server successfully respond a search query over the encryption? To address the problem, *searchable encryption (SE)* [28] has been introduced to allow a valid user to issue a search query, so that the server can return the corresponding encrypted data without knowing "what the query is" and "what the underlying data is".

In theory, SE schemes with no information leakage can be built on top of several well studied cryptographic primitives such as oblivious RAM, fully-homomorphic encryption, and secure two-party computation [32]. However, such constructions require either heavy computation complexity or large storage cost so that they may be inadequately employed in practice [6], [24]. Researchers have focused on designing searchable symmetric encryption (SSE) schemes with better efficiency (e.g., [10], [28]), which can be deployed in practice, but being subject to some information leak. Cash *et al.* [10] claimed that "in order to provide truly practical SSE solutions one needs to accept a certain level of leakage". One of the most recent concerns on SSE is the *leakage* of query [9]. One may aim to understand how much information an SSE scheme does truly leak in reality and meanwhile, what the information does mean for the scheme. Informally, there are two main types of information that may be leaked in practice. One is *search pattern* [9], [32] that can be used to determine if two queries correspond to the same keyword(s), and the other is *file-access pattern* [9], [32] referring to the information that is implied by the query results.

Recently, researchers have attempted to launch attacks against SSE schemes (those with some level of information leakage). Islam *et al.* [19] demonstrated that a server can identify the queries of a user from the leakage of search and file-access patterns if the server is with knowledge of all the contents of the user's files. Cash *et al.* [9] later presented an enhanced attack for larger keyword universe even if the server has less knowledge of the user's files. Liu *et al.* [23] proposed a new attack on keyword recovery, assuming the attacker has knowledge of the distribution on keyword(s) searched by user. Zhang *et al.* [32] introduced an "active" attack, called file-injection attack, in which the server is able to recover all the keywords from the given queries if it is allowed to actively inject some deliberately chosen files into the encrypted database. In this paper, we ask:

How effective are passive attacks against SSE schemes?

A. Our Contributions

In this paper, we study the prior knowledge and the search query leakage to quantitatively analyze the practical security

Manuscript received January 11, 2018; revised May 28, 2018 and July 19, 2018; accepted July 23, 2018. Date of publication August 21, 2018; date of current version September 13, 2018. This work was supported in part by the National Research Foundation, Prime Minister's Office, Singapore, under its Corporate Laboratory@University Scheme, National University of Singapore, and Singapore Telecommunications Ltd., and in part by the National Natural Science Foundation of China under Grant 61472357 and Grant 61571063. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Marina Blanton. *(Corresponding author: Jia Xu.)*

J. Ning and E.-C. Chang are with the Department of Computer Science, National University of Singapore, Singapore 117417 (e-mail: jtning88@gmail.com; changec@comp.nus.edu.sg).

J. Xu is with the NUS-Singtel Cyber Security Research and Development Laboratory, Singapore 117602 (e-mail: jia.xu@singtel.com).

K. Liang is with the Department of Computer Science, University of Surrey, Guildford GU2 7XH, U.K. (e-mail: k.liang@surrey.ac.uk).

F. Zhang is with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China, also with the Institute of Cyber Security Research, Zhejiang University, Hangzhou 310027, China, and also with the School of Computing, National University of Singapore 117417 (e-mail: fanzhang@zju.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2018.2866321

of SSE. We consider different amounts of prior knowledge that an attacker can obtain, and provide a series of passive attacks exploiting "*leakage of file-access pattern*". Different from the *count (or frequency) strategy*[1] used in [9] and [32], our attacks exploit the information of each keyword's "position" (i.e., the occurrence and non-occurrence of the keyword for each file). Our approach allow attackers to reveal more useful information which can be further used to launch keyword recovery attack. In this paper, we show that an attacker can reveal quite a high fraction of the underlying keywords corresponding to the tokens searched by users, with different amounts of prior knowledge. More details are described as follows.

- **Passive attack under the "full-fledged" assumption.** We first design a primary passive attack under the "full-fledged" assumption where the attacker has knowledge of the whole database and the encrypted file identifier (stored on the server) corresponding to each database file. This attack will be served as a "sub-routine" in the subsequent enhanced attacks.
- **Enhanced passive attack with no prior knowledge of file identifiers.** To relate adversary behaviours to more practical scenarios, we introduce a new assumption that requires the attacker to have no prior knowledge of the encrypted file identifiers (corresponding to each file). We note that the attacker here is not as powerful as the one under the "full-fledged" assumption. We further design an enhanced attack under this weaker assumption.
- **Passive attacks under other weaker assumptions.** We consider other weaker assumptions where the attacker has even less prior knowledge and design the enhanced passive attacks based on the assumptions.

We provide extensive experiments to display the success rate of our attacks. In each particular attack, the keyword recovery rate is close to the upper bound of the keyword recovery inherited by prior knowledge that the attacker has known.

B. Summary

We investigate the "power" of our passive attacks on current practical SSE. Our theoretical analysis and experimental results show that such (easy-to-mount) passive attacks can recover a significant amount of keywords embedded in search queries. In light of these, we recommend SSE designer to blur the (file identifier,token) inclusion/exclusion relationship, besides hiding the count/frequency of tokens.

C. Organization

We introduce preliminaries in Section II and the overview of assumptions in Section III. We start with a basic passive attack under the "full-fledged" assumption in Section IV. We further design an enhanced attack under a weaker assumption whereby the attacker has no prior knowledge of file identifiers in Section V. Based on the resulting attacks, we further introduce variants under other assumptions in Section VI. The experiment results of our attacks are presented in Section VII. Some discussions are given in Section VIII. Section IX discusses the related work, and Section X concludes this paper.

[1]The count/frequency strategy leverages the frequency of occurrence of the keywords (resp. tokens) in the known files (resp. the encrypted files stored on the server). Concretely, for a given keyword k and eight files, the count of k equals to 3 if there are 3 out of the files containing k.

II. Preliminaries and Problem Formulation

A. Preliminaries

Let $[n]$ denote a set $\{1, 2, ..., n\}$ of integers, and $A[1..n]$ denote a tuple $(A_1, A_2, ..., A_n)$, for $n \in \mathbb{N}$. For a set $\mathbf{S}$, let $|\mathbf{S}|$ denote the number of elements in $\mathbf{S}$. In general, an SSE scheme allows a user to securely outsource his/her encrypted database to a remote server, while the user still maintains the searchability over the encrypted data. Each file may be designed to contain a set of keywords such that the user could later retrieve the file by the keywords. Following [32], for simplicity, we regard a file as an unordered set of keywords. That is, a file is viewed as a set of keywords $\{k_1, k_2, ...\}$. If an attacker *knows* a file, it *knows* all the keywords of the file.

We briefly describe the search process of most current practical SSE schemes as follows. To retrieve a file (or files) from a server, a user *deterministically* constructs a search *token* t corresponding to a keyword k and further sends t to the server. Using t, the server searches the (encrypted) database so as to return the *file identifier(s)* of all the related file(s) containing k. With the identifier(s), the user obtains the file(s).

As noted in [9] and [32], since the token is deterministically derived from the keyword, the server will know the relationship between the "input" (i.e. token) and the "output" (i.e. file identifier). Given a token t, the server can identify the encrypted file(s) matching t. This is called *leakage of file-access pattern* [9], [32]. In this paper, we will design attacks on this type of leakage. The goal of our attacks is to reveal the relationship between the token and the underlying keyword and hence, violating the *query privacy*.[2] With knowledge of the recovered token (hereafter, by recovered token we mean the underlying keyword of the token is revealed), we can further violate the privacy of other "unknown" files, i.e., knowing the keywords of an "unknown" file. Concretely, for a query with a recovered token, we can identify if an unknown (encrypted) file contains the underlying keyword embedded into the token. The more tokens we recover, the more private information of the unknown files we can obtain.

B. Attack Model

1) Attacker: A server is the attacker who follows the specifications of SSE (which supports single keyword search) but curiously extracts as much secret information as possible.

2) Knowledge of Attacker: For different assumptions considered in this paper, an attacker may have the following knowledge.

- **Knowledge of (original) files (i.e. $\mathbf{M}_0$).** The attacker is allowed to obtain a set of files $\{F_i\}_{i\in[l]}$. Since the attacker knows $\{F_i\}_{i\in[l]}$, it can obtain the keyword(s) associated with each file in $\{F_i\}_{i\in[l]}$. That is, the attacker obtains a

[2]By violating the query privacy we mean that the attacker can obtain the underlying keyword of a given query.

set of files and the corresponding keywords. Assume the keyword set corresponding to $\{F_i\}_{i\in[l]}$ is $\{k_j\}_{j\in[n]}$. The attacker may generate an $l \times n$ matrix $\mathbf{M}_0$, where the row and column correspond to file and keyword, respectively. The value of the $(i, j)^{th}$ entry in $\mathbf{M}_0$ is defined as

$$m_{i,j} = \begin{cases} 0 & \text{if } k_j \notin F_i \\ 1 & \text{if } k_j \in F_i \end{cases}$$

The matrix $\mathbf{M}_0$ is set as

$$\begin{array}{c} \\ F_1 \\ \vdots \\ F_l \end{array} \begin{array}{c} \begin{array}{cccc} k_1 & k_2 & \cdots & k_n \end{array} \\ \begin{pmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ \vdots & \vdots & \cdots & \vdots \\ m_{l,1} & m_{l,2} & \cdots & m_{l,n} \end{pmatrix} \end{array}. \quad (1)$$

- **Knowledge of file identifiers and the corresponding tokens from observed search results (i.e. $\mathbf{M}_1$).** Let $\{F'_i\}_{i\in[l]}$ be the file identifier set of encrypted versions of a file set $\{F_i\}_{i\in[l]}$ stored on the server and $\{t_i\}_{i\in[n]}$ be the token set corresponds to $\{F'_i\}_{i\in[l]}$ (i.e., $\{F_i\}_{i\in[l]}$).[3] The server will recover an $l \times n$ matrix $\mathbf{M}_1$ from the observed search results, where the row and column correspond to file identifier and token, respectively. The value of the $(i, j)^{th}$ entry in $\mathbf{M}_1$ is defined as

$$m'_{i,j} = \begin{cases} 0 & \text{if } t_j \notin F'_i \\ 1 & \text{if } t_j \in F'_i \end{cases}$$

The matrix $\mathbf{M}_1$ is designed as

$$\begin{array}{c} \\ F'_1 \\ \vdots \\ F'_l \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & \cdots & t_n \end{array} \\ \begin{pmatrix} m'_{1,1} & m'_{1,2} & \cdots & m'_{1,n} \\ \vdots & \vdots & \cdots & \vdots \\ m'_{l,1} & m'_{l,2} & \cdots & m'_{l,n} \end{pmatrix} \end{array}. \quad (2)$$

- **Knowledge of the file identifier of each (known) file (i.e. π_r).** The attacker may know the file identifier of encrypted version of each (known) file. That is, the attacker knows the row permutations (denoted by π_r) between $\mathbf{M}_0$ and $\mathbf{M}_1$.

3) Objective of Attacker: The goal of the attacker is to reveal the relationship between keyword and token to compromise the underlying keyword of each queried token.

C. Problem Definition

Let $\mathbf{M}_0 = \begin{pmatrix} \vdots \\ \vec{r}_i \\ \vdots \end{pmatrix}_{i\in[l]}$ be a $l \times n$ matrix, where the row vector $\vec{r}_i$ denote its i-th row. Let π_r be a permutation over $[l]$. We define a matrix transformation function ROWPERMUTE as below

$$\text{ROWPERMUTE}(\mathbf{M}_0, \pi_r) = \begin{pmatrix} \vdots \\ \vec{r}_{\pi_r(i)} \\ \vdots \end{pmatrix}_{i\in[l]}, \quad (3)$$

where the i-th row of the resulting matrix will equal to the $\pi_r(i)$-th row of the input matrix $\mathbf{M}_0$.

Alternatively, we may write the matrix $\mathbf{M}_0$ in column vector form: $\mathbf{M}_0 = (\cdots, \vec{c}_j^{\mathrm{T}}, \cdots)_{j\in[n]}$, where the column vector $\vec{c}_j^{\mathrm{T}}$ denotes its j-th column. Let π_c be a permutation over $[n]$. We define a matrix transformation function COLPERMUTE as below

$$\text{COLPERMUTE}(\mathbf{M}_0, \pi_c) = \left(\cdots, \vec{c}^{\mathrm{T}}_{\pi_c(j)}, \cdots\right)_{j\in[n]}, \quad (4)$$

where the j-th column of the resulting matrix will equal to the $\pi_c(j)$-th column of the input matrix $\mathbf{M}_0$.

Similar to [9] and [19], we define the *recovery rate* as the percentage of unique queries correctly guessed (or recovered). That is, the recovery rate reflects the percentage of the unique (keyword,token) mappings that could be recovered.

The problem can be formalized as follows.

Problem 1: Alice randomly chooses two permutations $\pi_r : [l] \to [l]$ and $\pi_c : [n] \to [n]$. Let $\mathbf{M}_0$ be an $l \times n$ matrix. Alice computes the transformed matrix $\mathbf{M}_1 :=$ COLPERMUTE(ROWPERMUTE($\mathbf{M}_0, \pi_r$), π_c)). Then Alice sends the two matrix $(\mathbf{M}_0, \mathbf{M}_1)$ to Bob. Bob attempts to find the two permutation π_r, π_c. More precisely, Bob deterministically *outputs two arrays* MATCHEDROWS[1..l] *and* MATCHEDCOLS[1..n]*, such that for any $i \in [l]$,* MATCHEDROW[i] $\in \{\pi_r(i), \perp\}$*, and for any $j \in [n]$,* MATCHEDCOLS[j] $\in \{\pi_c(j), \perp\}$*, in order to maximize the recovery rate $\rho := \frac{|\{\text{MATCHEDCOLS}[j]\neq\perp\}|}{n}$.*

We remark that, in the above definition of Problem 1, we allow *false negative* result (i.e. in Bob's output, MATCHEDROWS[i] or MATCHEDCOLS[j] could be $\perp$ for some row index i, or column index j, respectively), but not accept *false positive* result (i.e.if MATCHEDROWS[i] $\notin \{\pi_r(i), \perp\}$ or MATCHEDCOLS[j] $\in \{\pi_c(j), \perp\}$ in Bob's output, then Bob's recovery rate is zero.)

Example 1: Let $\mathbf{M}_0$ be a matrix with every entry equals 0. Thus, after row permutations and column permutations, the resulting matrix is still a zero-matrix. That is $\mathbf{M}_1 = \mathbf{M}_0$. In this case, any permutation π_0 over $[l]$ and any permutation π_1 over $[n]$ can satisfy the equation $\mathbf{M}_1 =$ COLPERMUTE(ROWPERMUTE($\mathbf{M}_0, \pi_0$), π_1)*. Since there are exponentially many such permutations π_0, π_1, the probability that Bob will correctly guess the exact permutation π_r, π_c chosen by Alice, will be always less than 1 (actually, the probability is negligibly small, close to zero). As a result, Bob's recovery rate will be zero.*

Example 2: Let $l \geq n$ and $\mathbf{M}_0$ be a $l \times n$ matrix, such that for each column index j, in the j-th column of $\mathbf{M}_0$, there are exactly j number of 1 and $(l - j)$ number of 0. In this case, Bob can find the column level permutation π_c with certainty by counting number of 1's in each column in $\mathbf{M}_0$ and $\mathbf{M}_1$.

Lemma 1: For any (computationally unbounded) algorithm executed by Bob in Problem 1, the recovery rate cannot exceed the fraction of unique columns in matrix $\mathbf{M}_0$, that is,

$$\frac{|\{\text{MATCHEDCOL}[j] \neq \perp\}|}{n} \leq \frac{|\{\textit{Unique columns in } \mathbf{M}_0\}|}{n} \quad (5)$$

[3]Note that since (we assume) tokens are deterministically computed from the corresponding keywords, the size of the token set equals to the size of the keyword set.

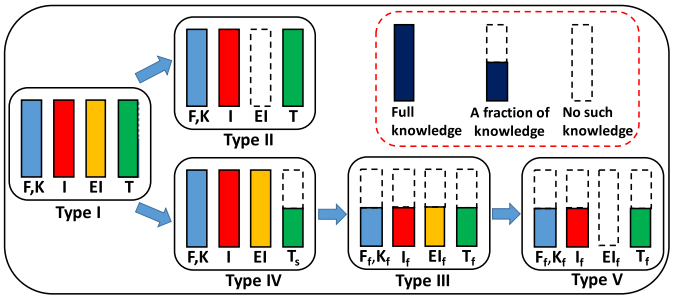


Fig. 1. Relationships and comparison among the five assumptions.[4]

Remark: We say a column in matrix $\mathbf{M}_0$ is *unique*, if its column vector is distinct from any other column vector in matrix $\mathbf{M}_0$. As indicated by Lemma 1, the upper bound on the recovery rate equals the fraction of unique columns in matrix $\mathbf{M}_0$. We note that the upper bound on the recovery rate does not consider the case where an attacker can correctly guess the permutation used among the non-unique columns.

III. Overview of Assumptions

In this paper, we mainly consider SSE with single-keyword search whereby one search token is only embedded with one keyword. The attack goal is to reveal the relationship between a chosen keyword and a given search token, i.e., compromising the underlying keyword of the token. We assume that the queries issued by users could be observed by the attacker in a reasonable amount of time, and the queries should exhaustedly cover all keywords under each assumption.

We first present a primary passive attack under the "full-fledged" assumption. The assumption may be a rare case in real-world applications from an attacker perspective. However, it could be seen as a worst case that the prior knowledge of attacker is maximized in passive attack. In this paper, we use this worst case as a start point of research. It is worth mentioning that the attack algorithm under the worst case assumption can be further served as a "subroutine" in the subsequent enhanced attacks.

In total, we mainly present five attacks under the respective assumptions in Section IV, Section V and Section VI. We introduce the assumptions below.

- **Type I assumption.** We assume the attacker is given all database files $\mathbf{F}$ and the corresponding keyword set $\mathbf{K}$.[5] The attacker is also able to relate the encrypted file identifier (stored on the server) with each file in $\mathbf{F}$ (i.e., **I**, **EI** in Fig. 1). In addition, the attacker is allowed to observe all the queries to associate the (returned) file identifier(s) with each search token, where the queries completely cover all keywords in $\mathbf{K}$ (i.e., it knows the token set $\mathbf{T}$ corresponding to $\mathbf{K}$). In other words, the attacker has knowledge of $l \times n$ matrices $\mathbf{M}_0$, $\mathbf{M}_1$ and the row permutation π_r (as defined in Section II), where $l = |\mathbf{F}|$ and $n = |\mathbf{K}|$. Since being given full knowledge of database, the attacker under the assumption is "full-fledged". Therefore, Type I assumption is referred to as the "full-fledged" assumption.
- **Type II assumption.** We assume the attacker is provided the same amount of knowledge as the one under the Type I assumption except that it is unable to link the file identifier (stored on the server) with each file. Likewise, the attacker has knowledge of $\mathbf{M}_0$ and $\mathbf{M}_1$.
- **Type III assumption.** We assume the attacker is given a fraction of $\mathbf{F}$ (denoted by $\mathbf{F_f}$) and the keyword set $\mathbf{K_f}$ corresponding to $\mathbf{F_f}$. It also knows the file identifier (stored on the server) of each file in $\mathbf{F_f}$ and the token set (denoted by $\mathbf{T_f}$) corresponding to $\mathbf{K_f}$. In addition, the attacker can observe all the queries to associate the (returned) file identifier(s) with each search token in $\mathbf{T_f}$, where the queries completely cover all keywords in $\mathbf{K_f}$. Concretely, the attacker has knowledge of $l' \times n'$ matrices $\mathbf{M}_0$, $\mathbf{M}_1$ and the row permutation π_r, where $l' = |\mathbf{F_f}| < |\mathbf{F}|$, $n' = |\mathbf{K_f}| < |\mathbf{K}|$.
- **Type IV assumption.** We assume the attacker is provided the same amount of knowledge as the one under the Type I assumption except that it only knows a fraction of $\mathbf{T}$ (i.e., $\mathbf{T_s}$ in Fig. 1).
- **Type V assumption.** We assume the attacker is given $\mathbf{F_f}$ and the keyword set $\mathbf{K_f}$. The attacker knows the file identifier set $\mathbf{I_f}$ for $\mathbf{F_f}$ and can further obtain the token set $\mathbf{T_f}$ for $\mathbf{K_f}$ by observing the (searched) tokens w.r.t. $\mathbf{F_f}$, where the queries of search tokens completely cover all keywords in $\mathbf{K_f}$. Therefore, the attacker has knowledge of $l' \times n'$ matrices $\mathbf{M}_0$ and $\mathbf{M}_1$, where $l' = |\mathbf{F_f}| < |\mathbf{F}|$, $n' = |\mathbf{K_f}| < |\mathbf{K}|$.

The relationships and comparison among these assumptions (which are organized in a hierarchy of decreasing adversarial background knowledge) are shown in Fig. 1. We use the following example to illustrate the practical scenario of each assumption. We consider a four-party setting of searchable encryption: Data owner (which is a company) has a dataset D (e.g. it could be healthcare data and human genome database), and encrypts it as D′ using some searchable encryption scheme before outsourcing it to a cloud server. Meanwhile, the data owner may authorize some data consumer (who could be a biomedical researcher) to make query over D′, and the cloud server is not supposed to be able to learn nor understand the dataset or queries. The fourth party is an inside attacker Eve, who is an employee in the company. Here D is the intellectual property of the data owner, and the sequence of queries themselves are intellectual property of the data consumer (e.g. two consecutive queries, which are a particular disease name and a particular human gene segment, may suggest that the data consumer is investigating the linkage between the disease and the gene segment, which might be a very valuable information for the competitors of the data consumer). We further note that in general a stringent security requirement of a secure system

[4] $\mathbf{F}$ is the set of all database files, $\mathbf{K}$ is the keyword set corresponding to $\mathbf{F}$, $\mathbf{F_f}$ is a fraction of $\mathbf{F}$, $\mathbf{K_f}$ is the keyword set corresponding to $\mathbf{F_f}$, $\mathbf{I}$ is the file identifier set of $\mathbf{F}$ (i.e., $\mathbf{I} = \{\mathbf{id}(\mathbf{x}) : \mathbf{x} \in \mathbf{F}\}$, where $\mathbf{id}(\cdot)$ is a function that maps a file to its file identifier), $\mathbf{I_f}$ is the file identifier set of $\mathbf{F_f}$ (i.e., $\mathbf{I_f} = \{\mathbf{id}(\mathbf{x}) : \mathbf{x} \in \mathbf{F_f}\}$), $\mathbf{EI}$ is the set of relationship between file identifier and file in $\mathbf{F}$ (i.e., $\mathbf{EI} = \{(\mathbf{x}, \mathbf{id}(\mathbf{x})) : \mathbf{x} \in \mathbf{F}\}$), $\mathbf{EI_f}$ is the relationship set for $\mathbf{F_f}$ (i.e., $\mathbf{EI_f} = \{(\mathbf{x}, \mathbf{id}(\mathbf{x})) : \mathbf{x} \in \mathbf{F_f}\}$), $\mathbf{T}$ is the token set for $\mathbf{K}$, $\mathbf{T_f}$ is the token set for $\mathbf{K_f}$, and $\mathbf{T_s}$ is a subset of $\mathbf{T}$.

[5] Recall that a file is a set of keywords. If an attacker knows a file, it knows all the keywords of the file.

(e.g. in leakage resilient cryptography) is that there should be *no leakage amplification*. In other words, after some out-of-band leakage of plaintext and some metadata, as long as the master private key is still completely random from the attacker, the attacker should not know *more* information of the plaintext. All of our attacks under Type I to V assumptions, in fact, demonstrate leakage amplification of plaintext.

- **Case 1 (Type I assumption):** Eve steals information **F**, **K**, **I**, **EI** from the data owner, and wants to know about the sequence of queries by the data consumer. So Eve colludes with the cloud server, who can monitor all queries made by the data consumer (i.e. **T**).
- **Case 2 (Type II assumption):** Similar to **Case 1**, except that Eve failed to steal information **EI**.
- **Case 3 (Type IV assumption):** Similar to **Case 1**, except that the cloud server is only able to obtain a subset (i.e., $\mathbf{T_s}$) of the whole token set after observing queries from the data consumer (i.e., not all keywords are searched).
- **Case 4 (Type III assumption):** Eve only steals information $\mathbf{F_f}$, $\mathbf{K_f}$, $\mathbf{I_f}$, $\mathbf{EI_f}$. For example, D′ is dynamic in the sense that the data owner collects new data and outsources it to the cloud server every week. Eve succeeded in stealing $\mathbf{F_f}$, $\mathbf{K_f}$, $\mathbf{I_f}$, $\mathbf{EI_f}$ only in a particular week. Eve also colludes with the cloud server, who could obtain $\mathbf{T_f}$.
- **Case 5 (Type V assumption):** Similar to **Case 4**, except that Eve failed to steal $\mathbf{EI_f}$.

IV. Passive Attack Under Type I Assumption

In this section, we present a primary passive attack under the "full-fledged" assumption. Note that allowing attacker to have knowledge of file identifier is a practical assumption [32]. This is because: 1) the attacker can identify the returned (encrypted) file based on its length (in reality, it may be inadequately to always pad each file to the maximum file length); 2) for SSE schemes supporting update functionality, the attacker can compromise a file and identify the file with the next encrypted file uploaded by user.

The problem (to be tackled) under the "full-fledged" assumption is a simplified version of the one defined in Subsection II-C. The attack mainly exploits the information of each keyword's "position" (i.e., the occurrence and non-occurrence of the keyword for each file). For example, given a particular keyword k and the eight files (which are already exposed to the attacker), we define a binary sequence (01001010) indicating that file no. 2, file no. 5 and file no. 7 contains k. Such a binary sequence is a reflection of the "position" of k in a particular set of files. Our attack leverages the uniqueness of such "position" information to recover the underlying keywords of search queries. Specifically, we view each column of $\mathbf{M}_0$ and $\mathbf{M}_1$ as binary sequence. The binary sequence in each column of $\mathbf{M}_0$ is a reveal of matching relation between a particular keyword and the (whole) files that the attacker has known. Likewise, the sequence in each column of $\mathbf{M}_1$ is a reveal of matching relation between a particular token and the (whole) file identifiers (that the attacker has known).

The basic observation is that if we can map each column of $\mathbf{M}_0$ with that of $\mathbf{M}_1$, we can launch an attack that could map each keyword with the corresponding token. Different from the count (or frequency) method in [9] and [32], the novelty of our attack lies in the usage of the uniqueness of each column from matrices $\mathbf{M}_0$ and $\mathbf{M}_1$, where the uniqueness depends on the binary sequence of each column (rather than simply counting the number of file that appears). Since we are considering the one-(keyword)-to-one-(token) mapping, given a unique column from $\mathbf{M}_0$ with binary sequence c, there must exist a unique column from $\mathbf{M}_1$ whose binary sequence exactly matches c.

Cornerstone of Attack. Leverage the uniqueness of each column to establish the mapping between keyword (i.e., column of $\mathbf{M}_0$) and its corresponding token (i.e., column of $\mathbf{M}_1$).

Description of Attack. The attack consists of two steps:

- **Convert:** Convert the binary sequence of each column of $\mathbf{M}_0$ and $\mathbf{M}_1$ to an integer, and obtain two sets of integers, COLINT_0 for $\mathbf{M}_0$ and COLINT_1 for $\mathbf{M}_1$.
- **Match:** For each integer k that is unique among COLINT_0, find t in $\mathbf{M}_1$ satisfies: (1) t is unique among COLINT_1; (2) $t = k$. Let K be the underlying keyword that k represents and T be the underlying token that t represents, output each (keyword,token) mapping (K, T).

The pseudocode of this attack and the corresponding algorithm analysis are given in Appendix A. There exists an upper bound on the recovery rate of this attack. Under the "full-fledged" assumption, $\mathbf{M}_1$ is indeed a "half-permuted" version of $\mathbf{M}_0$ in the sense that the columns of $\mathbf{M}_1$ are the permutations of the columns of $\mathbf{M}_0$. The fundamental task of this attack is to identify the "pre-image" (in $\mathbf{M}_0$) of each column of $\mathbf{M}_1$. Intuitively, for columns of $\mathbf{M}_0$ (or $\mathbf{M}_1$), the percentage of unique column is the upper bound of the (keyword,token) mappings that we can recover. Therefore, the keyword recovery rate equals to this upper bound, that is, the recovery rate of our attack is optimal.

V. Enhanced Passive Attack Under Type II Assumption

Relating file identifier with each file stored on server is indeed a strong assumption that allows the attacker to obtain sufficient knowledge of the database. In practice, an attacker may have no prior knowledge of the file identifier corresponding to each file at all. In this section, we consider an assumption that is weaker than the "full-fledged" one. In particular, the assumption is identical to that of Section IV except that the attacker cannot relate the file identifier with each (known) file.

Without prior knowledge of file identifiers, we cannot reuse the passive attack under the "full-fledged" assumption directly. We here need an "extra-processing" step. The main task of the step is to obtain (keyword,token) and (file,file identifier) mappings as many as possible. The following two methods are leveraged to find the mappings. The first is the count strategy that is used to map keyword (resp. file) with token (resp. file identifier) if they share the same unique count. The second method we use, is the *count track*.

For each keyword (resp. file), the count track is defined to be the vector of counts corresponding to different sets of files (resp. keywords). For example, given a particular keyword k and eight files, the count is 6 if there are six out of the files containing k; for a subset of all the files, say five of them, the count is 3 if there are three out of the five files containing k. The count track related to k is $\{6, 3\}$. The mapping between keyword (resp. file) and token (resp. file identifier) relies on the same unique count track number they have.

A. Main Idea of Attack

Our approach is to find some features or characteristics which could serve as "fingerprint" to identify different columns (or rows) of a matrix:

- such features should stay constant even if the rows and columns of a matrix are randomly permutated;
- different columns (or rows, respectively) are likely to have different feature values.

In particular, the attack under this weak assumption consists of two steps. The first is to use the idea of *Count and Match* to obtain (file,file identifier) and (keyword,token) mappings as many as possible. The second is to utilize the recovered (file,file identifier) mappings (obtained from the first step) to launch a similar attack as given in Section IV to recover the "more" unique (keyword,token) mappings.

B. Algorithm Description

Since the second step of the attack is identical to the attack in Section IV, we here mainly give the algorithm description of the first step. We will sum all entries in each column of $\mathbf{M}_0$ and $\mathbf{M}_1$. If for some $j_0, j_1 \in [n]$, the following conditions hold:

- Sum of the j_0-th column of $\mathbf{M}_0$ is unique among all sums of each column of $\mathbf{M}_0$;
- Sum of the j_1-th column of $\mathbf{M}_1$ is unique among all sums of each column of $\mathbf{M}_1$;
- Sum of the j_0-th column of $\mathbf{M}_0$ is equal to the sum of the j_1-th column of $\mathbf{M}_1$,

we will identify (j_0, j_1) as a matched pair and set $\text{MATCHEDCOL}[j_1] = j_0$. Next, we conceptually remove the j_0-th column from $\mathbf{M}_0$ by setting each entry in this column to 0, and conceptually remove the j_1-th column from $\mathbf{M}_1$ by setting each entry in this column to 0. In a similar way, we might match some row (say i_0-th row) of $\mathbf{M}_0$ with some row (say i_1-th row) of $\mathbf{M}_1$, and conceptually remove them by setting corresponding matrix entries to 0.

After these two matrix are updated, we will calculate sums for each column/row of each matrix again. As a result, in every matrix, each column/row will be associated with a tuple of two sums. If for some $j_2, j_3 \in [n]$, the following conditions hold:

- The tuple[6] of sums for the j_2-th column of $\mathbf{M}_0$ is unique among all tuples of sums for each column of $\mathbf{M}_0$;
- The tuple of sums for the j_3-th column of $\mathbf{M}_1$ is unique among all tuples of sums of each column of $\mathbf{M}_1$;
- The tuple of sums for the j_2-th column of $\mathbf{M}_0$ equals the tuple of sum for the j_3-th column of $\mathbf{M}_1$,

we can identify (j_2, j_3) as a matched pair and set $\text{MATCHEDCOL}[j_3] = j_2$. Again, we *conceptually*[7] remove these two columns from the two matrix correspondingly. We apply similar procedures over rows, identify matched rows from these two matrix, and eventually conceptually remove these matched rows. We can recursively perform the above procedures, until we cannot find more matched columns or rows.

The pseudocode of this attack and the corresponding algorithm analysis are given in Appendix B. $\mathbf{M}_1$ is indeed a "full-permuted" version of $\mathbf{M}_0$ in the sense that both the columns and rows of $\mathbf{M}_1$ are the permutations of the columns and rows of $\mathbf{M}_0$, respectively. The fundamental task of the attack is to identify the "pre-image" (in $\mathbf{M}_0$) of each column of $\mathbf{M}_1$. For columns of $\mathbf{M}_0$, the percentage of unique columns is the upper bound of the (keyword,token) mappings that we can recover, which is inherited by the knowledge of the attacker under the Type II assumption.

VI. Passive Attacks Under Other Weaker Assumptions

The attacks under the "full-fledged" and Type II assumptions effectively compromise the underlying keywords of queries. In practice, however, attackers may not have so much prior knowledge. In this section, we introduce several variants of the attacks in Section IV and V based on new assumptions.

A. Attack Under Type III Assumption

The "full-fledged" assumption supposes the attacker knows all the database files. In practice, however, an attacker may not be given such an advantage. We here consider a weaker but more reasonable assumption. It is identical to the "full-fledged" assumption except that the attacker only has partial knowledge of all files. Concretely, the attacker is provided: 1) a fraction of all database files, $\mathbf{F_f}$, and the corresponding keyword set $\mathbf{K_f}$; 2) the file identifier (stored on the server) for each file in $\mathbf{F_f}$ (i.e., $\mathbf{I_f}$, $\mathbf{EI_f}$ in Fig. 1); 3) the token set $\mathbf{T_f}$ for $\mathbf{K_f}$. An attacker can observe the *query-response* pairs and collect all distinct queries (i.e. the tokens), the responses (i.e., search results) of which contain at least one file identifier in $\mathbf{I_f}$. We assume that the attacker can obtain the entire token set $\mathbf{T_f}$ for $\mathbf{K_f}$ after observing for a reasonable amount of time.[8] With the above knowledge, the attacker can obtain $|\mathbf{F_f}| \times |\mathbf{K_f}|$ matrices $\mathbf{M_0}$, $\mathbf{M_1}$ and the row permutation π_r. Since $\mathbf{M_0}$ and $\mathbf{M_1}$ share the same dimensions and the attacker knows the file identifier corresponding to each file in $\mathbf{F_f}$, we can make use of the same attack method introduced in Section IV to recover the mapping of (keyword,token).

[6]It is possible that, neither the first sum nor the second sum for the j_2-th column is unique, but the tuple of two sums could be unique.

[7]We do not actually delete any rows or columns, in order to keep the dimension of the two matrix unchanged, and thus all row/column indices will not change.

[8]The limitation of this assumption will be discussed in Section VIII.

B. Attack Under Type IV Assumption

We here consider another variant that is identical to the "full-fledged" assumption except that the adversary is given a subset of the token set associated with the known files. Specifically, the attacker is provided: 1) the whole database $\mathbf{F}$ and the corresponding keyword set $\mathbf{K}$; 2) the file identifier (stored on the server) of each file in $\mathbf{F}$ (i.e., $\mathbf{I}$, $\mathbf{EI}$ in Fig. 1); 3) a subset of all the tokens corresponding to $\mathbf{K}$, denoted by $\mathbf{T_s}$. Therefore, the attacker can obtain an $|\mathbf{F}| \times |\mathbf{K}|$ matrix $\mathbf{M_0}$ and an $|\mathbf{F}| \times |\mathbf{T_s}|$ matrix $\mathbf{M_1}$, where $|\mathbf{K}| > |\mathbf{T_s}|$.

1) Cornerstone of Attack: The observation here is that the row dimension of $\mathbf{M_0}$ is the same as that of $\mathbf{M_1}$. For a unique column $\vec{c}_0$ from $\mathbf{M_0}$, if there exists a column $\vec{c}_1$ from $\mathbf{M_1}$ that shares the same binary sequence with $\vec{c}_0$, then $\vec{c}_1$ is unique among all columns of $\mathbf{M_1}$. One can establish the mapping between the underlying keyword of $\vec{c}_0$ and the underlying token of $\vec{c}_1$. The attack is to find the mapping of $(\vec{c}_0, \vec{c}_1)$ as many as possible.

2) Description of Attack: The attack consists of two steps:

- **Convert:** Convert the binary sequence of each column of $\mathbf{M}_0$ and $\mathbf{M}_1$ to an integer, and further obtain two sets of integers, COLINT_0 for $\mathbf{M}_0$ and COLINT_1 for $\mathbf{M}_1$.
- **Match:** For each integer k that is unique among COLINT_0, try to find t in $\mathbf{M}_1$ satisfies: (1) t is unique among COLINT_1; (2) $t = k$. If such t exists, let K be the underlying keyword that k represents and T be the underlying token that t represents, output each (keyword, token) mapping (K, T).

Let $\mathbf{T}$ denote the token set corresponding to $\mathbf{K}$ and $\mathbf{M}'_1$ be an $|\mathbf{I}| \times |\mathbf{T}|$ matrix, where the row and column correspond to file identifier in $\mathbf{I}$ and token in $\mathbf{T}$, respectively. Let $\mathbf{C}$ be the set of columns from $\mathbf{M}_1$, where each element of $\mathbf{C}$ is unique among all the columns of $\mathbf{M}'_1$. Intuitively, for columns of $\mathbf{M}_1$, the percentage of columns in $\mathbf{C}$ is the upper bound of the mappings that we can recover. Therefore, the keyword recovery rate equals to this upper bound, which is optimal.

C. Attack Under Type V Assumption

The Type II assumption supposes that the attacker is given $\mathbf{F}$. We here consider a more reasonable assumption. The knowledge of the attacker is the same with the Type III assumption, excepting that it can only know the file identifier set (denoted by $\mathbf{I_f}$) corresponding to $\mathbf{F_f}$ (i.e., no knowledge of $\mathbf{EI_f}$). We note that providing $\mathbf{I_f}$ for the attacker is reasonable, because: 1) the attacker can identify the returned files based on their lengths (e.g., the lengths of the encrypted files of $\mathbf{F_f}$ are all within a certain range); 2) In some cases, the attacker may compromise a set of files (which is to be uploaded by a user) via various of means, so that it can identify the file identifier set corresponding to $\mathbf{F_f}$ (with the uploaded encrypted files). The attacker can obtain an $|\mathbf{F_f}| \times |\mathbf{K_f}|$ matrix $\mathbf{M_0}$ and an $|\mathbf{I_f}| \times |\mathbf{T_f}|$ matrix $\mathbf{M_1}$, where $|\mathbf{F_f}| = |\mathbf{I_f}|$, $|\mathbf{K_f}| = |\mathbf{T_f}|$. Since $\mathbf{M_0}$ and $\mathbf{M_1}$ share the same dimensions and the attacker knows $\mathbf{I_f}$, we can make use of the same attack method used in Section V to recover the mapping of (keyword, token).

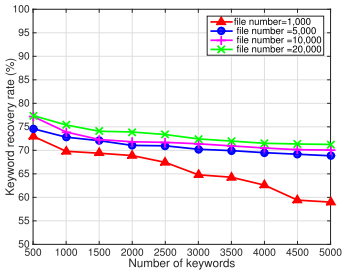


Fig. 2. Keyword recovery rate under the "full-fledged" assumption.

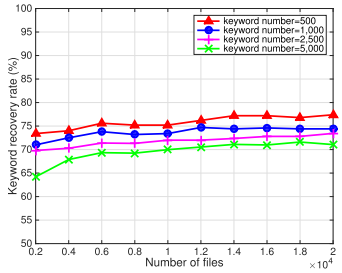


Fig. 3. Keyword recovery rate under the "full-fledged" assumption.

VII. EXPERIMENTS

In this section, we simulate the attacks proposed in Section IV, V and VI. Since our assumptions are different from but also weaker than that of [32], we do not compare our attacks with those introduced in [32].

A. Setting

Similar to [9] and [32], we adopt the Enron email dataset [1], which consists of 30,109 emails from 150 users of the Enron corporation from 2000-2002. We remove the headers (e.g., from, to, date), punctuations and numbers. We then use the standard Porter stemming algorithm [27] to process each word, and remove stop words (such as the, to, in, for, and) from the keyword list. We choose the top 5,000 most frequent keywords as the keyword universe (as that of [9] and [32]). In our experiments, we treat one email message as one file.

Since we cannot find a real-world query dataset for email, much like [9], [32], in our experiments, the queries of a user are chosen uniformly from the keyword universe.[9] Similarly, leaked files (i.e., the files known to the attacker) are chosen uniformly from the file universe, i.e., the set of 30,109 emails. For each case of attack, we repeat each instance of attack 20 times and eventually take the average. As defined in Section IV, the recovery rate is the percentage of unique queries correctly recovered. In the following experiments, we first fix the number of files and evaluate the keyword recovery rate for varying number of distinct keywords queried from 500 to 5,000.

[9]This means each (distinct) keyword will be chosen with the same probability, so that the case where very few distinct keywords are searched is (almost) impossible to happen.

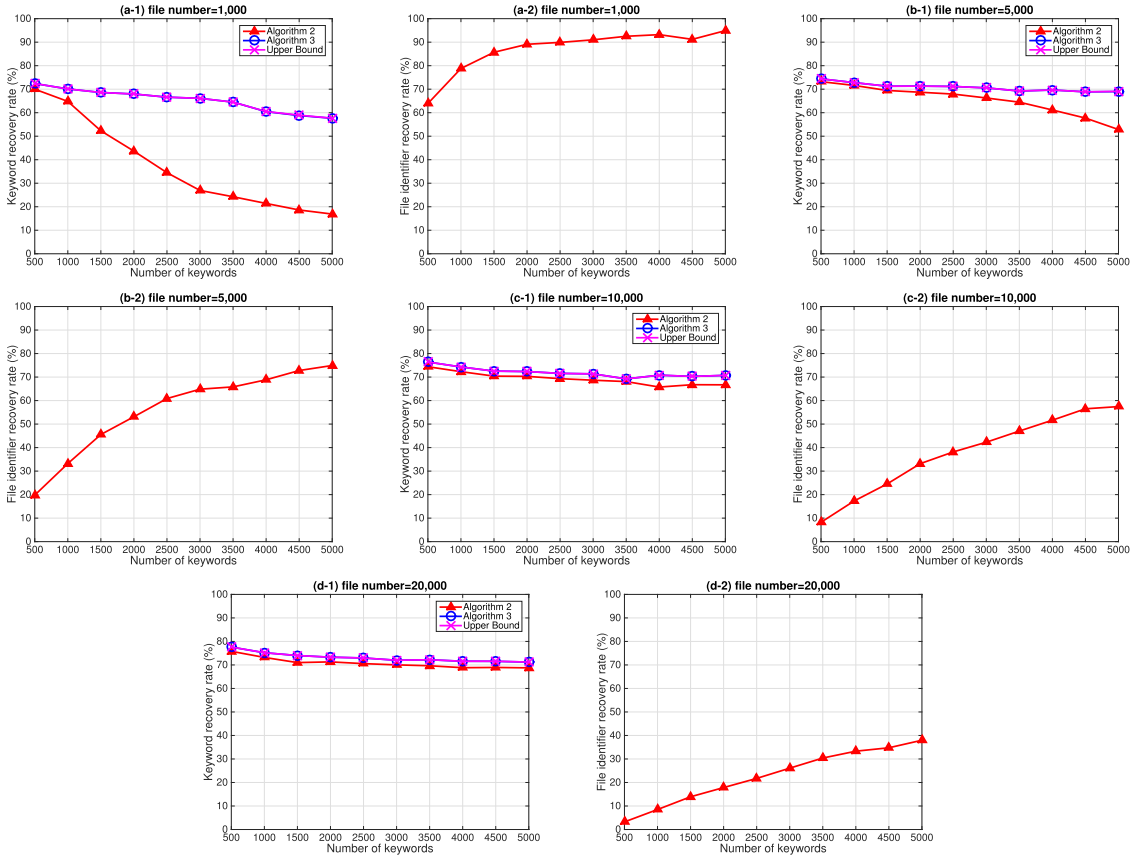


Fig. 4. Keyword/File identifier recovery rate of the attack (Algorithm 2 and 3) under Type II assumption. We remark that in sub-figures (a-1), (b-1), (c-1) and (d-1), the curve of Algorithm 3 and the curve of the upper bound almost coincide.

B. Recovery Under the "Full-Fledged" Assumption

We implement our attack under the "full-fledged" assumption. In our experiments, we target the keyword recovery of our attack in different scenarios. Fig. 2 shows that for four different cases (i.e., when the number of files are 1,000, 5,000, 10,000 and 20,000), our attack can recover about 70% (on average) of the keywords which have been queried. We then fix the number of keywords and evaluate the keyword recovery rate for varying number of files from 2,000 to 20,000. Fig. 3 shows that for four different cases (i.e., when the number of distinct keywords queried are 500, 1,000, 2,500 and 5,000), our attack can recover about 72% (on average) of the keywords which have been queried. It can be observed that the attack under the "full-fledged" assumption performs quite well. We emphasize that the recovery rate of this attack is optimal.

C. Recovery Under Type II Assumption

The recovery rate of the attack under Type II assumption is shown in Fig. 4. We report the keyword recovery of Algorithm 2, Algorithm 3 and the corresponding upper bound of keyword recovery, respectively. In addition, we record the file identifier recovery of Algorithm 2. In sub-figures (a-1), (b-1), (c-1), (d-1), there are three curves showing how the keyword recovery rate changes with the number of keywords w.r.t. a fixed number of files. Among the 3 curves, one represents the keyword recovery rate of Algorithm 2, one for Algorithm 3, and the third curve represents the upper bound of keyword recovery rate. The (corresponding) file identifier recovery of each experiment (i.e., when file number equals 1,000, 5,000, 10,000, 20,000) is shown in sub-figures (a-2), (b-2), (c-2), (d-2), respectively. We emphasize that, the curve for Algorithm 3 is almost coincided with the curve for the upper bound, which demonstrates that our Algorithm 3 achieves almost optimal recovery rate.

D. Recovery Under Type III Assumption

The attack under Type III assumption is implemented here. Fig. 5 shows that for four different cases (i.e., when the percentage of files are 10%, 20%, 50% and 80%), our attack can recover about 65%∼70% of the keywords have been queried. In addition, we fix the number of distinct keywords queried (for four different cases: 500, 1,000, 2,500 and 5,000) and evaluate the keyword recovery rate for varying percentage of files from 10% to 80%. Fig. 6 shows that our attack can recover about 68%∼72% (on average) of the keywords which have been queried. It can be observed that the attack performs quite well, where the recovery rate is optimal.

E. Recovery Under Type IV Assumption

For the attack under Type IV assumption, we fix the number of files (for two different cases: 5,000 and 10,000) and evaluate

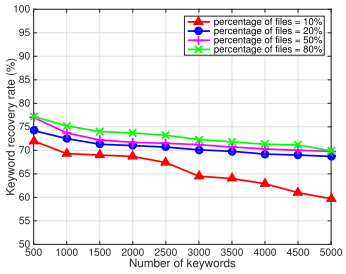


Fig. 5. Keyword recovery rate of the attack under Type III assumption.

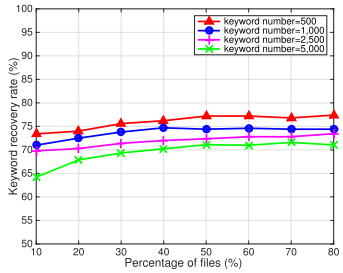


Fig. 6. Keyword recovery rate of the attack under Type III assumption.

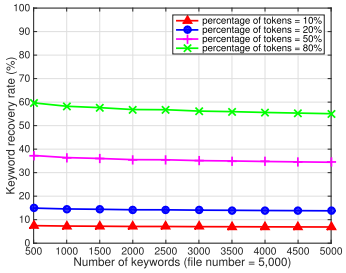


Fig. 7. Keyword recovery rate of the attack under Type IV assumption.

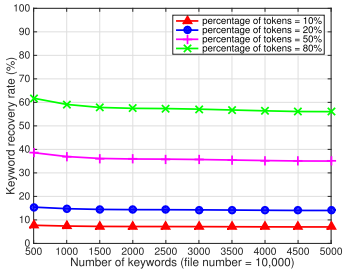


Fig. 8. Keyword recovery rate of the attack under Type IV assumption.

the keyword recovery rate for varying number of distinct keywords queried from 500 to 5000. Fig. 7 and Fig. 8 show the keyword recovery rates of the attack for four different cases (i.e., when the percentage of tokens are 10%, 20%, 50% and 80%) when the number of files equals 5,000 and 10,000, respectively.

F. Recovery Under Type V Assumption

Fig. 9 shows the keyword recovery rate of our attack under Type V assumption. For four cases (i.e., when the percentage of files are 10%, 20%, 50% and 80%), our attack can recover about 68%∼71% (on average) of the keywords which have been queried. Our experiments indicate that the proposed attack is quite effective. Since the assumption of this attack is very close to the one introduced in [9], we compare our attack under Type V assumption with the one in [9] (denoted as CGPR15) in Fig. 10. In the experiment, the percentage of files denotes the files known to the attacker, and the top 2,000 most frequent keywords are considered. As shown in Fig. 10, our attack outperforms [9] w.r.t. the keyword recovery rate.

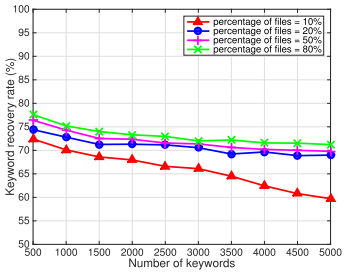


Fig. 9. Keyword recovery rate of the attack under Type V assumption.

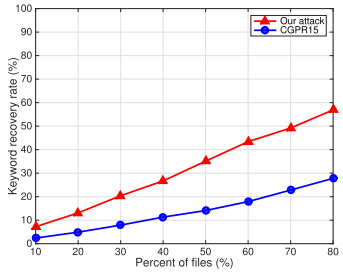


Fig. 10. Comparison of keyword recovery rate.

VIII. Discussions

A. Limitations of the Attacks

The effectiveness of the present attacks in this paper mainly depends on knowledge an attacker obtains from observing file access pattern. We assume that the attacker is able to obtain the matrix $\mathbf{M_1}$. In some special cases, the attacker may only get a subset of the columns of $\mathbf{M_1}$ (those corresponding to the keywords actually queried) over a certain/fixed period of time, so that the proposed attacks may not be effective. Besides, if the attacker only has a subset of the columns of $\mathbf{M_0}$, the keyword recovery rate might be (much) lower than that of the attack under "full-fledged" assumption (with knowledge of all columns of $\mathbf{M_0}$). For the assumptions in Subsection VI-A and Subsection VI-C, we suppose that the attacker can obtain the entire token set $\mathbf{T_f}$ corresponding to $\mathbf{K_f}$ after observing the files in $\mathbf{F_f}$. In practice, the attacker possibly may need to observe for a very long time to achieve this.

B. Possible Countermeasures

In order to protect SSE schemes against the attacks on file access patten, researchers [9], [19] have considered to leverage *keyword padding* as a countermeasure. The premise of the technique is to add noise to distort the real frequency (or count) of keyword. In an SSE scheme with keyword padding, given a query token of a keyword k, the returned search result will

Algorithm 1 Passive Attack Under the "Full-Fledged" Assumption

Input: An $l \times n$ matrix $\mathbf{M}_0 = (m_{i,j})_{i\in[l],j\in[n]}$ and an $l \times n$ matrix $\mathbf{M}_1 = (m'_{i,j})_{i\in[l],j\in[n]}$, $m_{i,j}, m'_{i,j} \in \{0, 1\}$.
Output: MATCHEDCOLS$[1..n]$, where MATCHEDCOLS$[j] = k$ means the k-th keyword corresponds to the j-th token.

Initialize three arrays with length n: MATCHEDCOLS$[j] := \perp$, COLINT$_0[j] := \perp$, COLINT$_1[j] := \perp$, $j = 1, 2, 3, \ldots, n$.
Initialize two sets GOODCOLS$_0$, GOODCOLS$_1$ as empty sets.
for each column index $j \in [n]$ **do**
  $t := \sum_{i=1}^{l} m_{i,j} \cdot 2^{i-1}$ ▷ Convert the binary sequence of the j-th column of $\mathbf{M}_0$ to an integer
  COLINT$_0[j] = t$
  $t' := \sum_{i=1}^{l} m'_{i,j} \cdot 2^{i-1}$ ▷ Convert the binary sequence of the j-th column of $\mathbf{M}_1$ to an integer
  COLINT$_1[j] = t'$
end for

Compute GOODCOLS$_0 = \{j \in [n]$: the value COLINT$_0[j]$ is unique among COLINT$_0[1..n]\}$
Compute GOODCOLS$_1 = \{j \in [n]$: the value COLINT$_1[j]$ is unique among COLINT$_1[1..n]\}$
for each column index $j \in$ GOODCOL$_0$ **do**
  Try to find k such that COLINT$_1[k] ==$ COLINT$_0[j]$
  if such k exists and $k \in$ GOODCOLS$_1$ **then**
    Let MATCHEDCOL$[k] := j$
  end if
end for
return MATCHEDCOLS.

include all the files containing k along with additional noised files that are not embedded with k. However, trivially making use of the padding may not be applicable to our attacks. Since we consider an SSE scheme in the single-keyword search setting, the number of returned files is padded up to the same length (for each keyword) for each query token. That is, for the matrix $\mathbf{M}_0$, the number of 1 corresponding to the binary sequence of each column might always equal to the same integer. The count (or frequency) attack may fail since the number of returned files is padded to the point that no keyword has a unique counting. But the binary sequences of columns may still maintain distinct with high probability. Therefore, our attacks may stand still under keyword padding countermeasure. We here suggest that an effective solution, w.r.t. our attacks, should aim to scramble the uniqueness of each sub-sequence of the binary sequence corresponding to each matrix column.

IX. Related Work

There are two main research branches of provably secure SE [6]: *searchable SE (SSE)* and *public key encryption with keyword search (PEKS)*. Many SSE schemes have been designed to support various features of searchability: 1) single keyword search [15], [25], [28]; 2) fuzzy keyword search [21], [22], [30]; 3) conjunctive keyword search [10], [13], [18]; and 4) ranked and verifiable keyword search [8], [11], [29], [31]. PEKS, which is mainly based on public key encryption, also have been studied to support: 1) single keyword search [2], [3]; 2) conjunctive keyword search [5], [20]; 3) fuzzy keyword search [4], [7]; and 4) verifiable keyword search [33].

In theory, one can construct SE schemes with no information leakage based on several well-known techniques such as fully-homomorphic encryption [14], [17], secure two-party computation [26], and oblivious RAM [16]. However, such schemes are impractical [6], [24]. Researchers focused on designing efficient SSE schemes (e.g., [12], [25], [28]) that can be efficiently implemented in real-world applications, at the expense of allowing the leakage of some information. Liu *et al.* [23] presented two attacks that can reveal the underlying keywords of queries by utilizing some public available knowledge. Islam *et al.* [19] demonstrated that the leakage of search and file-access patterns (together with some other knowledge) can be used to reveal the underlying keywords of queries. Cash *et al.* [9] later proposed an attack for larger keyword universe with less knowledge than that of [19]. Recently, Zhang *et al.* [32] presented a file-injection attack whereby the attacker can actively inject some files deliberately chosen by the attacker. In this work, we design some attacks on SSE supporting single keyword search, without making use of additional public available information and considering file injection.

X. Conclusion

This paper investigates the vulnerability of SSE schemes based on the assumptions on how much prior knowledge an attacker is allowed to achieve. Several passive attacks, which only using passive knowledge to reveal the relationship between keyword and search token, are introduced in the paper. The experiments demonstrate that the passive attacks can be launched to a special type of SSE schemes, which fail to "hide" the file-access pattern, to effectively break query privacy. This work also inspires researchers to pay attention on balancing the trade-off between efficiency and information leakage in the design of SSE schemes.

Appendix A
Pseudocode of Passive Attack Under the "Full-Fledged" Assumption and Its Analysis

The attack is described in the pseudocode of Algorithm 1. We further give the analysis of the algorithm as follows.

Algorithm 2 First Step of Enhanced Passive Attack Under Type II Assumption

Input: An $l \times n$ matrix M_0 and an $l \times n$ matrix M_1.

Output: MATCHEDCOLS$[1..n]$ and MATCHEDROWS$[1..l]$, where MATCHEDCOLS$[j] = k$ means the k-th keyword corresponds to the j-th token, and MATCHEDROWS$[i] = k$ means the k-th file identifier corresponds to the i-th file.

1: Initialize two arrays: MATCHEDROWS$[i] := \perp$, $i = 1, 2, 3, \ldots, l$; MATCHEDCOLS$[j] := \perp$, $j = 1, 2, 3, \ldots, n$.
2: Initialize four arrays: ROWSUM$_0[i]$, ROWSUM$_1[i]$, $i = 1, 2, 3, \ldots, l$; COLSUM$_0[j]$, COLSUM$_1[j]$, $j = 1, 2, 3, \ldots, n$.
3: Initialize four sets GOODROWS$_0$, GOODROWS$_1$, GOODCOLS$_0$, GOODCOLS$_1$ as empty sets.
4:
5: **loop**
6: **for** each row index $i \in [1, l]$ **do**
7: $t :=$ *sum of all matrix entry* $M_0[i][j]$ *in* i*-th row* ▷ Some matrix entries may be updated in previous loop
8: ROWSUM$_0[i]$.*append*(t) ▷ Insert value t from the tail of vector ROWSUM$[i]$
9: $t :=$ *sum of all matrix entry* $M_1[i][j]$ *in* i*-th row* ▷ Some matrix entries may be updated in previous loop
10: ROWSUM$_1[i]$.*append*(t) ▷ Insert value t from the tail of vector ROWSUM$_1[i]$
11: **end for**
12:
13: **for** each column index $j \in [1, n]$ **do**
14: $t :=$*sum of all matrix entry* $M_0[i][j]$ in j-th column ▷ Some matrix entries may be updated in previous loop
15: COLSUM$_0[j]$.*append*(t) ▷ Insert value t from the tail of vector COLSUM$[j]$
16: $t :=$ *sum of all matrix entry* $M_1[i][j]$ *in* j*-th column*
17: COLSUM$_1[j]$.*append*(t) ▷ Insert value t from the tail of vector COLSUM$[j]$
18: **end for**
19:
20: Compute NEWGOODROWS$_0 :=$ {$i \in [1, l] \backslash$GOODROWS$_0$ s.t. the value ROWSUM$_0[i]$ is unique among ROWSUM$_0[1..l]$}
21: **for** each $k \in$ NEWGOODROWS$_0$ **do**
22: Try to find i such that ROWSUM$_1[i] ==$ ROWSUM$_0[k]$
23: **if** such i exists and the value ROWSUM$_1[i]$ is unique among ROWSUM$_1[1..l]$ **then**
24: Add i into the set GOODROWS$_1$ and let MATCHEDROWS$[i] := k$
25: Set every entry in k-th row of matrix $\mathbf{M}_0$ to 0
26: Set every entry in i-th row of matrix $\mathbf{M}_1$ to 0
27: **end if**
28: **end for**
29: Update the set GOODROWS$_0 :=$ GOODROWS$_0 \cup$ NEWGOODROWS$_0$ ▷ Note set GOODROWS$_1$ has been updated already
30:
31: Compute NEWGOODCOLS$_0 :=$ {$j \in [1, n] \backslash$GOODCOLS$_0$ s.t. the value COLSUM$_0[j]$ is unique among COLSUM$_0[1..n]$}
32: **for** each $k \in$ NEWGOODCOLS$_0$ **do**
33: Try to find j such that COLSUM$_1[j] ==$ COLSUM$_0[k]$
34: **if** such j exists and the value COLSUM$_1[j]$ is unique among COLSUM$_1[1..n]$ **then**
35: Add j into the set GOODCOL$_1$ and let MATCHEDCOLS$[j] := k$
36: Set every entry in k-th column of matrix $\mathbf{M}_0$ to 0
37: Set every entry in j-th column of matrix $\mathbf{M}_1$ to 0
38: **end if**
39: **end for**
40: Update the set GOODCOLS$_0 :=$ GOODCOLS$_0 \cup$ NEWGOODCOLS$_0$ ▷ Note set GOODCOLS$_1$ has been updated already
41:
42: **if** both NEWGOODROWS$_0$ **and** NEWGOODCOLS$_0$ are empty sets **then**
43: **terminate the outermost loop**
44: **end if**
45: **end loop**
return MATCHEDROWS and MATCHEDCOLS.

Claim 1: If $\mathbf{M}_1 =$ COLPERMUTE$(\mathbf{M}_0, \pi_c)$ *for some permutation* π_c *over* $[n]$*, then at the end of the first* ***for*** *loop of Algorithm 1 (i.e. Line 9), the following equations hold*

$$\forall j \in [n], \quad \text{COLINT}_1[j] = \text{COLINT}_0[\pi_c(j)] \tag{6}$$

Proof: After columns of matrix $\mathbf{M}_0$ are permutated according to π_c, the $\pi_c(j)$-th column of matrix $\mathbf{M}_0$ will become the j-th column of matrix $\mathbf{M}_1$. Each column vector does not change, and only their positions have changed. The value COLINT$_1[j]$ (or COLINT$_0[\pi_c(j)]$, respectively) is only

dependent on the j-th (or $\pi_c(j)$-th, respectively) column vector of matrix $\mathbf{M}_1$ (or $\mathbf{M}_0$, respectively), regardless of the positions of columns. □

Claim 2: If $\mathbf{M}_1 = \textsc{ColPermute}(\mathbf{M}_0, \pi_c)$ *for some permutation* π_c *over* $[n]$*, then at the end of Algorithm 1, the following conditions hold*

$$\forall j \in [n], \quad \textsc{MatchedCol}[j] \in \{\pi_c(j), \perp\} \tag{7}$$

Proof: If $\textsc{MatchedCol}[j] \neq \perp$, Line 15 of Algorithm 1 is executed: $\textsc{MatchedCol}[k] := j$. Therefore, all of pre-conditions of Line 15 hold: (1) $\textsc{ColInt}_0[j]$ is unique among $\textsc{ColInt}_0[1..n]$; (2) $\textsc{ColInt}_1[k]$ is unique among $\textsc{ColInt}_1[1..n]$; (3) $\textsc{ColInt}_1[k] == \textsc{ColInt}_0[j]$. Combining these 3 conditions with Claim 1, Claim 2 is implied. □

Appendix B
Pseudocode of Passive Attack in Section V and Its Analysis

The attack is presented in the pseudocode of Algorithm 2 and 3 and the analysis of the algorithm is given as follows.

Define function $f : \mathbb{N}^{l\times n} \to \mathbb{N}^{l\times 1}$ as below

$$f(\mathbf{M}_0) = f\left(\begin{pmatrix} \vdots \\ \vec{r}_i \\ \vdots \end{pmatrix}_{i\in[l]}\right) = \begin{pmatrix} \vdots \\ \sum_{j=1}^{n} r_{i,j} \\ \vdots \end{pmatrix}_{i\in[l]}, \tag{8}$$

where vector $\vec{r}_i = (r_{i,1}, \ldots, r_{i,j}, \ldots, r_{i,n})$.

Claim 3: For any matrix $\mathbf{M}$ *and any permutation* π_c *over* $[n]$ *and any permutation* π_r *over* $[l]$*,*

$$f(\textsc{ColPermute}(\mathbf{M}, \pi_c)) = f(\mathbf{M}) \tag{9}$$

$$f(\textsc{RowPermute}(\mathbf{M}, \pi_r)) = \textsc{RowPermute}(f(\mathbf{M}), \pi_r). \tag{10}$$

Assumption 1: In Algorithm 2, the two input matrix $\mathbf{M}_0$ *and* $\mathbf{M}_1$ *satisfy this relationship:* $\mathbf{M}_1 = \textsc{ColPermute}(\textsc{RowPermute}(\mathbf{M}_0, \pi_r), \pi_c))$ *for some permutation* π_r *over* $[l]$ *and some permutation* π_c *over* $[n]$*.*

In the loop of Algorithm 2, both $\mathbf{M}_0$ and $\mathbf{M}_1$ could be updated by setting some entries to zero. For ease of exposition, let $\mathbf{M}_0^{(\zeta)}$ denote the $\mathbf{M}_0$ at the end of the ζ-th loop iteration, and $\mathbf{M}_1^{(\zeta)}$ denote the matrix $\mathbf{M}_1$ at the end of the ζ-th loop iteration. In addition, let $\mathbf{M}_0^{(0)}$ and $\mathbf{M}_1^{(0)}$ refer to the input matrix $\mathbf{M}_0$ and $\mathbf{M}_1$, respectively. It is easy to see that, during the execution of Algorithm 2, in the data structures $\textsc{MatchedRows}$, $\textsc{MatchedCols}$, $\textsc{RowSums}_0$, $\textsc{ColSums}_0$, $\textsc{RowSums}_0$, and $\textsc{ColSums}_0$, the value of any variable (e.g. array element, vector element) will not be overwritten, that is, once a variable is assigned to some value other than special symbol $\perp$, this variable will not be assigned to any other value again. In addition, the values of these variables at the beginning of $(\zeta+1)$-th iteration, are identical to those at the end of ζ-th iteration, $\zeta \in [N-1]$, where N is the total number of iterations before the loop terminates.

Lemma 2: If Assumption 1 holds and the loop terminates after N *iterations, for every integer* $\zeta \in [N]$*, then at the end of* ζ*-th iteration of the loop in Algorithm 2 (i.e. Line 45 of Algorithm 2), the following conditions hold:*

$$\forall i \in [l], \quad \textsc{RowSum}_1[i][1..\zeta] = \textsc{RowSum}_0[\pi_r(i)][1..\zeta] \tag{11}$$

$$\forall j \in [n], \quad \textsc{ColSum}_1[j][1..\zeta] = \textsc{ColSum}_0[\pi_c(j)][1..\zeta] \tag{12}$$

$$\forall i \in [l], \quad \textsc{MatchedRows}[i] \in \{\pi_r(i), \perp\} \tag{13}$$

$$\forall j \in [n], \quad \textsc{MatchedCols}[j] \in \{\pi_c(j), \perp\} \tag{14}$$

$$\mathbf{M}_1^{(\zeta)} = \textsc{ColPermute}(\textsc{RowPermute}(\mathbf{M}_0^{(\zeta)}, \pi_r), \quad \pi_c)) \tag{15}$$

Proof of Lemma 2: It is easy to see that, in Algorithm 2: a) Equation (11)(12)(13)(14)(15) hold trivially at the beginning of the 1st iteration; b) Values of all variables at the beginning of some iteration of the main loop, are identical to the values of those variables at the end of previous iteration. Based on the above two simple facts, Lemma 2 can be directly implied by the following Lemma 3. □

Lemma 3: Let N *be specified as in Lemma 2. Let* $\zeta \in [N]$*. If Equation* (11)(12)(13)(14)(15) *hold at the very beginning of the* ζ*-th loop iteration, then these equations also hold at the end of the* ζ*-th loop iteration.* Note that the dimension of the vector $\textsc{RowSum}[i]$ (or $\textsc{ColSum}[j]$, respectively) is $(\zeta-1)$ at the beginning and is ζ at the end of the ζ-th loop iteration.

Proof of Lemma 3 for Equation (11) *and Equation* (12)*:* At the end of the ζ-th loop, the dimension of $\textsc{RowSum}_0[i]$ increases by 1, and a new element $\textsc{RowSum}_0[i][\zeta]$ is appended from the tail. According to Algorithm 2, $\textsc{RowSum}_0[i][\zeta]$ is the sum of all entries in the i-th row of $\mathbf{M}_0^{(\zeta-1)}$. Thus, at the end of ζ-th loop

$$\begin{pmatrix} \vdots \\ \textsc{RowSum}_0[i][\zeta] \\ \vdots \end{pmatrix}_{i\in[l]} = f(\mathbf{M}_0^{(\zeta-1)}) \tag{16}$$

Similarly,

$$\begin{pmatrix} \vdots \\ \textsc{RowSum}_1[i][\zeta] \\ \vdots \end{pmatrix}_{i\in[l]} = f(\mathbf{M}_1^{(\zeta-1)}) \tag{17}$$

By applying homomorphism of function f, we have

$$f(\mathbf{M}_1^{(\zeta-1)}) \tag{18}$$

$$= f(\textsc{ColPermute}(\textsc{RowPermute}(\mathbf{M}_0^{(\zeta-1)}, \pi_r), \quad \pi_c))) \tag{19}$$

$$= f((\textsc{RowPermute}(\mathbf{M}_0^{(\zeta-1)}, \pi_r))) \tag{20}$$

$$= \textsc{RowPermute}(f(\mathbf{M}_0^{(\zeta-1)}), \pi_r) \tag{21}$$

$$= \textsc{RowPermute}(\begin{pmatrix} \vdots \\ \textsc{RowSum}_0[i][\zeta] \\ \vdots \end{pmatrix}_{i\in[l]}, \pi_r) \tag{22}$$

$$= \begin{pmatrix} \vdots \\ \textsc{RowSum}_0[\pi_r(i)][\zeta] \\ \vdots \end{pmatrix}_{i\in[l]} \tag{23}$$

Algorithm 3 Enhanced Attack Under Type II Assumption

Input: An $l \times n$ matrix M_0 and an $l \times n$ matrix M_1

Output: MATCHEDCOLS$[1..n]$ and MATCHEDROWS$[1..l]$, where MATCHEDCOLS$[j] = k$ means the k-th keyword corresponds to the j-th token, and MATCHEDROWS$[i] = k$ means the k-th file identifier corresponds to the i-th file.

1: Invoke Algorithm 2 with input $(\mathbf{M}_0, \mathbf{M}_1)$, and obtain output MATCHEDROWS and MATCHEDCOLS.
2: Initialize a $l \times n$ matrix $\mathbf{M}_2$ with every entry equal to 0.
3: **for** each $i \in [l]$ **do**
4: let $i' :=$ MATCHEDROWS$[i]$
5: **if** $i' \neq \perp$ **then**
6: let the i'-th row of matrix $\mathbf{M}_2$ equals to the i-th row of matrix $\mathbf{M}_1$
7: **end if**
8: **end for**
9: **for** each $j \in [n]$ **do**
10: let $j' :=$ MATCHEDCOLS$[j]$
11: **if** $j' \neq \perp$ **then**
12: let every entry in the j'-th column of $\mathbf{M}_0$ and every entry in the j-th column of $\mathbf{M}_2$ be 0.
13: **end if**
14: **end for**
15: Invoke Algorithm 1 on the two matrix $\mathbf{M}_0$ and $\mathbf{M}_2$ to obtain $\overline{\text{MATCHEDCOLS}}$
return MATCHEDROWS and, the combination of MATCHEDCOLS and $\overline{\text{MATCHEDCOLS}}$

Combining Eq (17) and Eq (23), we have

$$\forall i \in [l], \quad \text{ROWSUM}_1[i][\zeta] = \text{ROWSUM}_0[\pi_r(i)][\zeta]. \quad (24)$$

Recall that, we assume Equation (11) holds at the beginning of the ζ-th loop iteration, that is, $\forall i \in [l]$,

$$\text{ROWSUM}_1[i][1..(\zeta-1)] = \text{ROWSUM}_0[\pi_r(i)][1..(\zeta-1)] \quad (25)$$

Combination of Eq (24) and Eq (25) leads to Eq (11) directly, as we desired. We can prove Eq (12) in a similar way, and thus save the details. □

Proof of Lemma 3 for Equation (13) *and Equation* (14)*:* It is easy to see that, during the execution of Algorithm 2, MATCHEDROW$[i]$ will be assigned to a value rather than $\perp$ for at most once. At the end of the ζ-th iteration, for any $i \in [l]$, there are exactly 3 possible cases which are mutually exclusive: 1) MATCHEDROW$[i] = \perp$; 2)MATCHEDROW$[i] \neq \perp$ at the beginning of the ζ-th iteration; 3) MATCHEDROW$[i]$ is assigned to some value i' ($i' \neq \perp$) during the ζ-th iteration.

If Case 1) occurs, then Equation (13) holds trivially at the end of ζ-th iteration. If Case 2) occurs, then by our assumption that MATCHEDROW$[i] \in \{\pi_r(i), \perp\}$ at the beginning of the ζ-th iteration, we can derive MATCHEDROW$[i] = \pi_r(i)$.

Now we handle the Case 3). In Case 3), Line 24 of Algorithm (2) has been executed at the ζ-th iteration. Therefore, the following 3 conditions hold:

- ROWSUM$_1[i] =$ ROWSUM$_0[k]$
- ROWSUM$_1[i]$ is unique among ROWSUM$_1[1..l]$
- ROWSUM$_0[k]$ is unique among ROWSUM$_0[1..l]$

The above 3 conditions and Equation (11) together lead to $\pi_r(i) = k$, i.e. MATCHEDROW$[i] = k = \pi_r(i)$ as desired. Consequently, Equation (13) is proved. Equation (14) can be proved in a similar way and we save the details. □

Proof of Lemma 3 for Equation (15)*:* Recall that a part of precondition of Lemma 3 is that Equation 15 holds at the beginning of the ζ-th iteration, i.e.

$$\mathbf{M}_1^{(\zeta-1)} = \text{COLPERMUTE}(\text{ROWPERMUTE}(\mathbf{M}_0^{(\zeta-1)}, \pi_r), \pi_c)) \quad (26)$$

From the Algorithm 2, we can see that entries of matrix $\mathbf{M}_0, \mathbf{M}_1$ are only updated

- in rows at Line 25 and Line 26, where MATCHEDROW$[i]$ is set to k and k-th row of matrix $\mathbf{M}_0$ and i-th row of matrix $\mathbf{M}_1$ are set to zero vectors;
- in columns at Line 36 and Line 37, where MATCHEDCOL$[j]$ is set to k, and k-th column of matrix $\mathbf{M}_0$, and j-th column of matrix $\mathbf{M}_1$ are set to zero vectors.

By Case 3) of our proof of Lemma 3 for Equation (13) and Equation (14), we have MATCHEDROW$[i] = k = \pi_r(i)$ and MATCHEDCOL$[j] = k = \pi_c(j)$ (Note that here the two occurrences of symbol k represents two independent temporary variables). Therefore, after this updating, the relationship in Equation (15) still holds. □

REFERENCES

[1] *Enron Email Dataset*. Accessed: Jun. 23, 2017. [Online]. Available: https://www.cs.cmu.edu/~./enron/

[2] M. Abdalla *et al.*, "Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions," *J. Cryptol.*, vol. 21, no. 3, pp. 350–391, Jul. 2008.

[3] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. EUROCRYPT*, 2004, pp. 506–522.

[4] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith, III, "Public key encryption that allows PIR queries," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2007, pp. 50–67.

[5] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. TCC*. Berlin, Germany: Springer, 2007, pp. 535–554.

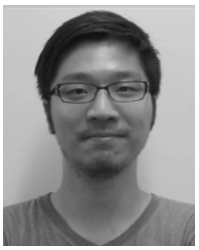
[6] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Comput. Surv.*, vol. 47, no. 2, 2015, Art. no. 18.

[7] J. Bringer, H. Chabanne, and B. Kindarji, "Error-tolerant searchable encryption," in *Proc. ICC*, Jun. 2009, pp. 1–6.

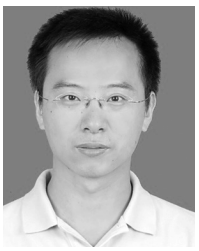
[8] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.
[9] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. ACM CCS*, 2015, pp. 668–679.
[10] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2013, pp. 353–373.
[11] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 917–922.
[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. ACM CCS*, 2006, pp. 79–88.
[13] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *Computer Security—ESORICS*. Berlin, Germany: Springer, 2015, pp. 123–145.
[14] K. Gai and M. Qiu, "Blend arithmetic operations on tensor-based fully homomorphic encryption over real numbers," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3590–3598, Aug. 2017.
[15] S. Gajek, "Dynamic symmetric searchable encryption from constrained functional encryption," in *Proc. Cryptographers' Track RSA Conf.* Berlin, Germany: Springer, 2016, pp. 75–89.
[16] S. Garg, P. Mohassel, and C. Papamanthou, "TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2016, pp. 563–592.
[17] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. STOC*, 2009, pp. 169–178.
[18] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. ACNS*. Berlin, Germany: Springer, 2004, pp. 31–45.
[19] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. NDSS*, vol. 20, 2012, p. 12.
[20] J. Lai, X. Zhou, R. H. Deng, Y. Li, and K. Chen, "Expressive search on encrypted data," in *Proc. Asia CCS*, 2013, pp. 243–252.
[21] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. INFOCOM*, Mar. 2010, pp. 1–5.
[22] C. Liu, L. Zhu, L. Li, and Y. Tan, "Fuzzy keyword search on encrypted cloud storage data with small index," in *Proc. IEEE Int. Conf. Cloud Comput. Intell. Syst. (CCIS)*, Sep. 2011, pp. 269–273.
[23] C. Liu, L. Zhu, M. Wang, and Y.-A. Tan, "Search pattern leakage in searchable encryption: Attacks and new construction," *Inf. Sci.*, vol. 265, pp. 176–188, May 2014.
[24] M. Naveed, "The fallacy of composition of oblivious RAM and searchable encryption," IACR Cryptol. ePrint Arch., Tech. Rep., 2015, p. 668. [Online]. Available: https://eprint.iacr.org/2015/668
[25] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 639–654.
[26] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Proc. ASIACRYPT*, 2009, pp. 250–267.
[27] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 40, no. 3, pp. 211–218, 2006.
[28] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, May 2000, pp. 44–55.
[29] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. ICDCS*, Jun. 2010, pp. 253–262.
[30] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *Proc. INFOCOM*, Mar. 2012, pp. 451–459.
[31] J. Wang, X. Chen, X. Huang, I. You, and Y. Xiang, "Verifiable auditing for outsourced database in cloud computing," *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3293–3303, Nov. 2015.
[32] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. USENIX Secur. Symp.*, 2016, pp. 707–720.
[33] Q. Zheng, S. Xu, and G. Ateniese, "VABKS: Verifiable attribute-based keyword search over outsourced encrypted data," in *Proc. INFOCOM*, Apr./May 2014, pp. 522–530.

Jianting Ning received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, in 2016. He is currently a Research Fellow with the Department of Computer Science, National University of Singapore. His research interests include applied cryptography and information security, in particular, public key encryption, attribute-based encryption, searchable encryption, and secure multi-party computation.

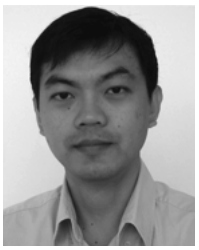
Jia Xu received the Ph.D. degree in computer science from the National University of Singapore in 2012. He is currently a Cyber Security Research and Development Manager specializing in cloud and data security with the NUS-SingTel Cyber Security Research and Development Laboratory. He was a Research Scientist with the Institute for Infocomm Research from 2012 to 2017. He is interested in applied cryptography and cloud computing security.



Kaitai Liang received the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, in 2014. He is currently an Assistant Professor with the Department of Computer Science, University of Surrey, U.K. His research interests are applied cryptography and information security in particular, encryption, network security, big data security, privacy-enhancing technology, and security in cloud computing.



Fan Zhang received the Ph.D. degree from the Department of Computer Science and Engineering, University of Connecticut, in 2012. He is currently an Associate Professor with the College of Information Science and Electronic Engineering, and also with the Institute of Cyber Security Research, Zhejiang University. He is also a Visiting Professor with the School of Computing, National University of Singapore. His research interests include system security, hardware security, cryptography, computer architecture, and sensor network.



Ee-Chien Chang received the B.Sc. and M.Sc. degrees from the National University of Singapore (NUS) and the Ph.D. degree from New York University, New York, in 1998. He was with the Department of Computational Science, NUS, in 1998 and 2000. He was a Post-Doctoral Fellow with DIMACS and the NEC Research Institute from 1999 to 2000. He is currently an Associate Professor with the Department of Computer Science, School of Computing, NUS. His research interests include network security and applied cryptography.