Access Pattern Hidden Query over Encrypted Data through Multi-clouds

Yi Dou and Henry C. B. Chan
Department of Computing, The Hong Kong Polytechnic University, Hong Kong
Email: yi.dou@connect.polyu.hk

***Abstract*—Searchable encryption seeks to support untrusted third parties to conduct direct searching over encrypted data. However, recent research has found that searchable encryption is vulnerable to attacks, which exploit the statistical relationship or pattern identified from encrypted query results. In this paper, we study the problem of access pattern leakage attack on searchable encryption under a multi-cloud environment. Basically, both database records and queries are distributed among different cloud servers, so that each cloud server can only have partial information about queries and their results. To minimize the query response time while protecting information disclosure, we formulate the record and query assignment as an optimization problem, and solve the problem (i.e., finding the best possible solution) by the minimum $s-t$ cut algorithm. Numerical results show that on average 13% access pattern information can be saved by our assignment strategy while maintaining good query response time.**

I. Introduction

Searchable encryption allows database searching to be conducted over encrypted data, which is particularly useful in the cloud environment. Basically, using the secure indexes generated by the data owner, a cloud server matches the encrypted query keywords (called trapdoor) with the secure indexes to find the required data. Currently, most Searchable Symmetric Encryption (SSE) schemes are developed based on the security model as defined by Curtmola et al. in [1]. However, this model can only safeguard the confidentiality of static data. During a query process, the cloud server can identify or know the encrypted record of a particular trapdoor query thus causing the access pattern leakage. In other words, if two trapdoors have very close query results, the corresponding query keywords are closely related. For instance, the keywords 'education' and 'university' may often appear together. By finding the co-occurrence probability of keywords in a plaintext dataset, the cloud server can identify the plaintext of trapdoor keywords [2]. Furthermore, it is possible to determine the trapdoor keywords based on the number of encrypted query results [3]. According to [4], over 50% of keywords have unique appearance frequencies in some datasets. Attackers can identify the trapdoor keywords based on the unique appearance frequencies in a plaintext dataset.

Oblivious RAM (ORAM) [5], [6] has been proposed to address the access pattern leakage problem by hiding the RAM (Random Access Memory) access patterns of executed programs from the server. However, the computational complexity of most ORAM schemes is high and they can only support limited types of queries. Similar to our approach, vertical fragmentation [7] seeks to use multiple servers to make attackers unable to discover the sensitive association between record attributes, (e.g., separating records of employee name and salary on two servers). But this technique cannot prevent the access pattern leakage attacks on a single attribute, and can lead to a long response time for multi-keyword search.

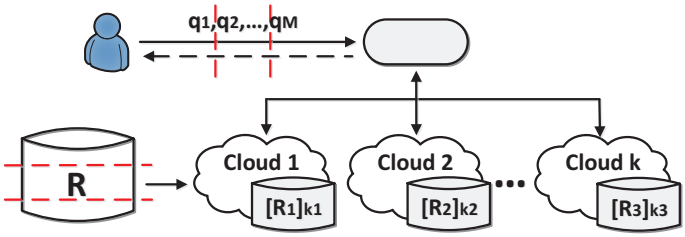


Fig. 1. Assigning records and queries among multiple clouds.

As shown in the Fig. 1, to tackle access pattern leakage attack, we distribute both database records (rows) and queries among different cloud servers. Since none of the servers can access the entire database of records and queries, they cannot identify the accurate statistical relationship of the query results. Before storing records to servers, all records are protected using the same SSE scheme, but each server is given different secret keys to encrypt and build the secure index for the records assigned to it. Accordingly, different trapdoors are created based on the server that is executing the query. Our scheme supports any secure SSE schemes.

We adopt the servers on multiple clouds, but in the same region. This deployment requires less bandwidth and reduces the chance for a colluding attack among servers (i.e., compared with deployment for servers on the different datacenters of the same cloud). To minimize both query response time and access pattern disclosure, we formulate the record and query assignment as an optimization problem. We then determine an optimal assignment strategy based on a minimum cut algorithm. In our experiments, we evaluated the assignment strategy output from our approach with a real world dataset. The numerical results indicate that on average 13% access pattern information can be saved by our assignment strategy, without sacrificing query response time. Furthermore, our algorithm is scalable under different realistic settings.

The remainder of the paper is organized as follows. Section

978-1-5090-5019-2/17/$31.00 ©2017 IEEE

II presents the security problem and strategies. Section III formulates our proposed record and query assignment model. Section IV describes how to search for an optimal assignment strategy by solving the assignment optimization problem. The experimental results and conclusion are shown in Section V and VI, respectively.

II. SECURITY STRATEGIES

In this section, we explain the reason for distributing both records and queries among multiple clouds by reviewing the previous access pattern leakage attacks.

A. Security Problems

1) IKK Attack: Islam, Kuzu, and Kantarcioglu (IKK) presented an attack that can recover the trapdoor keywords in the SSE schemes [2] based on the unique co-occurrence probability among keywords. An attacker has two matrices M_p and M_c. M_p is obtained from the public datasets (i.e., in plaintext) before searching. M_c is built by observing the trapdoors T_q (encrypted keywords), and their corresponding query results R_q (encrypted records) during SSE-based searching. Hence, the keywords are determined by matching elements between these two matrices. Each matrix element gives a co-occurrence probability of any two keywords or trapdoors appearing in the same record. For instance, (i,j)th element in M_c is $M_c(i,j) = \frac{|R_{q_i} \cap R_{q_j}|}{N}$, where N is the total number of encrypted records and $|R_{q_i} \cap R_{q_j}|$ is the number of elements in the intersection between result sets of query q_i and q_j. When the difference between $M_c(i,j)$ and $M_p(u,v)$ is close to zero, the attacker can map the ith and jth trapdoors in M_c to the plaintext keywords of the uth and vth queries in M_p.

2) Count Attack: Further to IKK attack, Cash et al. [3] presented another efficient attack that can discover trapdoor keywords. Basically, if an attacker has a complete encrypted record, some keywords can be identified by exploiting their unique frequency in the dataset. First, the attacker calculates the frequency of each keyword w in the public plaintext dataset (i.e., $count(w)$). Then during SSE searching, the server counts the number of records in each trapdoor query result, which gives the frequency of the trapdoor keyword in the encrypted dataset (i.e., $count(R_q)$). When a keyword in the plaintext dataset has the same or similar unique frequency as the trapdoor in SSE, that is $count(w) = count(R_q)$, the attacker can link the keyword w with the trapdoor T_q.

B. Security Strategies

Both the IKK and count attacks rely heavily on extracting the information from a complete plaintext dataset [8]. Hence an attacker cannot launch these attacks if only a fraction of the plaintext dataset is known [3]. In other words, if a server only knows partial information about the encrypted dataset, the above attacks can be minimized, even with the complete knowledge of the plaintext dataset.

1) Record Fragmentation: Both matrix element $M_c(i,j)$ and $count(R_q)$ are computed from the trapdoor query result set R_q. When the server is unable to access the correct query results of the trapdoors, the attacker finds it difficult to deduce their keywords by matching their frequencies. Padding is one of the approaches to hide actual query results, by adding dummy records and identifiers before building the secure indexes [2]. However, padding unsatisfied records causes false positives in the query results, and brings extra communication overhead [3], [8]. In this paper, we consider using multi-clouds to tackle the security problems. We will remove the records in the trapdoor query results to other cloud servers, in order to make the co-occurrence probability and keyword frequency observed by a single server incorrect. In other words, each server can only return part of the correct query results. The same query needs to be executed parallelly on multiple servers.

2) Query Distribution: Since IKK attack is based on the entire matrix M_c, the order of different query pairs co-occurrence probability can be used for keyword recovery attack. Apart from disturbing each query result set, hiding the differences between different $M_c(i,j)$ is also necessary. We adopt the same strategy proposed in [2]. Instead of using a padding approach, we will distribute queries with similar results on the same server, in order to make the query result of trapdoors on the same cloud server as similar as possible.

III. ASSIGNMENT STRATEGY

As shown in Fig. 1, the fundamental question is: "How should the records and queries be distributed among the multiple cloud servers in order to minimize the response time while satisfying certain security requirements?" In this section, we will introduce the strategy of record and query assignment, to satisfy the above introduced security requirements. Let $\mathcal{R} = \{r_1, ..., r_N\}$ denote a set of database records with several attributes. Let $\mathcal{Q} = \{q_1, ..., q_M\}$ denote a sequence of queries written in the development process. Each of the queries has different combinations of keywords to be searched on $\mathcal{R}$. In addition, let $\mathcal{S} = \{S_1, ..., S_k\}$ denote a group of cloud servers, which will be assigned both records and query tasks. Before distribution, the data owner builds a matrix $\mathcal{B}$ to describe the query results of all of the queries, in which the rows represent the record set and columns represent the query set. Each element $b_{r,q} \in \mathcal{B}$ is set to 1, if record $r \in \mathcal{R}$ satisfies query $q \in \mathcal{Q}$. The qth column vector of $\mathcal{B}$ denotes the query result of q, that is $\mathcal{B}_q$. Matrix $\mathcal{B}$ is safely kept by the data owner.

$$b_{r,q} = \begin{cases} 0, & \text{if } r \notin Result(q) \\ 1, & \text{if } r \in Result(q) \end{cases} \tag{1}$$

A. Record and Query Assignment

To prevent the aforementioned attacks, our scheme would assign both records and queries among cloud servers. We assume that the partial records and queries on a single cloud server cannot reveal the statistical property of the original access pattern.

Definition 1: **Record and Query Assignment:** We define $\mathcal{A} = (\mathcal{A}_R, \mathcal{A}_Q)$ as an assignment of records $\mathcal{R}$ and queries $\mathcal{Q}$

to a set of cloud servers $\mathcal{S}$. The process of assignment is to horizontally and vertically partition the elements of matrix $\mathcal{B}$ into $|\mathcal{S}|$ blocks. Finally, each cloud server obtains a sub-matrix of $\mathcal{B}$ with a subset of records and queries. To avoid repetition of the returned results, each element $b_{r,q} \in \mathcal{B}$ will be assigned to one of the cloud servers.

$$\mathcal{B} = \begin{array}{c} \begin{matrix} q_1 & q_2 & \dots & q_M \end{matrix} \\ \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 1 \end{pmatrix} \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_N \end{matrix} \rightarrow \begin{array}{c} \begin{matrix} q_1 & q_2 & \dots & q_M \end{matrix} \\ \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 1 \end{pmatrix} \end{array} \tag{2}$$

Matrix $\mathcal{A}_R$ describes the placement of records on cloud servers. Each element $x_{r,s} \in \mathcal{A}_R$ is set to 1, if record $r \in \mathcal{R}$ is assigned to cloud server $s \in \mathcal{S}$. Since each record is assigned to at least one cloud server, $\mathcal{A}_R$ has no zero row, that is $\sum_{s=1}^{s=k} x_{r,s} \geq 1$.

$$\mathcal{A}_R = \begin{array}{c} \begin{matrix} S_1 & \dots & \mathcal{A}_R^s & \dots & S_k \end{matrix} \\ \begin{pmatrix} x_{1,1} & \dots & x_{1,s} & \dots & x_{1,|\mathcal{S}|} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{|\mathcal{R}|,1} & \dots & x_{|\mathcal{R}|,s} & \dots & x_{|\mathcal{R}|,|\mathcal{S}|} \end{pmatrix} \end{array} \begin{matrix} r_1 \\ \vdots \\ r_N \end{matrix} \tag{3}$$

Matrix $\mathcal{A}_Q$ indicates the placement of queries on cloud servers. Each element $y_{q,s} \in \mathcal{A}_Q$ is set to 1, if query $q \in \mathcal{Q}$ is distributed to cloud server $s \in \mathcal{S}$. Each row vector of $\mathcal{A}_Q$ represents an assignment of a query among the cloud servers, that is $\mathcal{A}_Q^q = [y_{q,1}, ..., y_{q,|\mathcal{S}|}]$. Since each query must be distributed to at least one cloud server, $\mathcal{A}_Q$ has no zero row, that is $\sum_{s=1}^{s=k} y_{q,s} \geq 1$.

$$\mathcal{A}_Q = \begin{array}{c} \begin{matrix} S_1 & \dots & \mathcal{A}_Q^s & \dots & S_k \end{matrix} \\ \begin{pmatrix} y_{1,1} & \dots & y_{1,s} & \dots & y_{1,|\mathcal{S}|} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{q,1} & \dots & y_{q,s} & \dots & y_{q,|\mathcal{S}|} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{|\mathcal{Q}|,1} & \dots & y_{|\mathcal{Q}|,s} & \dots & y_{|\mathcal{Q}|,|\mathcal{S}|} \end{pmatrix} \end{array} \begin{matrix} q_1 \\ \vdots \\ \mathcal{A}_Q^q \\ \vdots \\ q_M \end{matrix} \tag{4}$$

We define the assignment on each cloud server s as $\mathcal{A}^s$, which is a tuple of two column vectors, that is $\mathcal{A}^s = (\mathcal{A}_R^s, \mathcal{A}_Q^s)$. Specifically, $\mathcal{A}_R^s = [x_{1,s}, ..., x_{|\mathcal{R}|,s}]^\intercal$ and $\mathcal{A}_Q^s = [y_{1,s}, ..., y_{|\mathcal{Q}|,s}]^\intercal$ correspond to the sth column in matrix $\mathcal{A}_R$ and $\mathcal{A}_Q$, respectively. It also means that each cloud server only obtains a subset of records and queries.

After applying the horizontal and vertical partition on matrix $\mathcal{B}$, we need to ensure records duplicated into multiple cloud servers cannot be associated. So, each cloud server reassigns different record IDs and uses different keys to encrypt records and their attribute names. Accordingly, different trapdoors are generated for the same query sent to different clouds. Finally, the independent searchable encryption scheme is applied within each cloud server.

B. Information Disclosure

In this subsection, we will clearly identify the information disclosure from the assignment $\mathcal{A}$ to the cloud server set $\mathcal{S}$. We assume that different cloud servers will not collude after applying different searchable encryption schemes. We firstly analyze the information disclosure on each single cloud server. By observing the response results of all of the queries, each cloud server s can rebuild another matrix $\mathcal{B}^s$ to describe the query results on its own. That is, for each $\forall x_{r,s} \in \mathcal{A}_R^s = 1$ and $\forall y_{q,s} \in \mathcal{A}_Q^s = 1$, the corresponding row and column vector in $\mathcal{B}$ is chosen to form $\mathcal{B}^s \subsetneq \mathcal{B}$ which is a sub-matrix of $\mathcal{B}$. The qth column vector of $\mathcal{B}^s$ denote the query result of q on the cloud s, that is $\mathcal{B}_q^s$.

$$\mathcal{B}^s = \begin{array}{c} \begin{matrix} \mathcal{A}_Q^s & y_{1,s} & y_{2,s} & \dots & y_{M,s} = 1 \end{matrix} \\ \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 1 \end{pmatrix} \end{array} \begin{matrix} \mathcal{A}_R^s \\ x_{1,s} \\ x_{2,s} \\ \vdots \\ x_{N,s} = 1 \end{matrix} \tag{5}$$

Definition 2: **Information Disclosure:** We define the information disclosure from the assignment $\mathcal{A}^s$ on each cloud server as a set of queries $D(\mathcal{A}^s)$ that does not satisfy the following constrains:

- Referring to the leakage definition proposed in [2], the result differences between any two queries on each cloud server s should be less than a threshold value $0 \leq \alpha \leq 1$. We use the Hamming distance between any two column vectors in $\mathcal{B}^s$ for evaluation. For any two queries $1 \leq \forall q_1, q_2 \leq cols(\mathcal{B}^s)$, if the
$$\frac{Hamming(\mathcal{B}_{q_1}^s, \mathcal{B}_{q_2}^s)}{rows(\mathcal{B}^s)} > \alpha \tag{6}$$
then q_1 and q_2 are regarded as insecure queries included to set $D(\mathcal{A}^s)$, that is $D(\mathcal{A}^s) \cup \{q_1, q_2\}$;
- None of the queries on the cloud server s should include the entire query result, that is, for $1 \leq \forall q \leq cols(\mathcal{B}^s)$, if $\mathcal{B}_q^s = \mathcal{B}_q$, then $D(\mathcal{A}^s) \cup \{q\}$.

We define the overall information disclosure of an assignment $\mathcal{A}$ as the ratio of insecure queries to the total queries.

$$D(\mathcal{A}) = \frac{\left| \bigcup_{s=1}^{s=k} D(\mathcal{A}^s) \right|}{|\mathcal{Q}|} \tag{7}$$

C. Query Response Time

We use the query response time observed by the data user to evaluate the performance of records and queries distribution across multiple cloud servers. We consider that all of the queries $q_1, ..., q_M$ have already been identified by the developers. And each query has a corresponding execute frequency, denoted as a vector $\mathcal{F} = [f_1, ..., f_M]$, $\sum_{q=1}^{q=M} f_q = 1$. The response time of each cloud server consists of two parts: local processing time and result transmission time. Since queries are sent to a cloud server in sequential order, the local processing of each cloud server can be modelled as a $M/M/1$ queue.

The time cost of query result transmission is a constant calculated based on the result size and network bandwidth, that is $Tran(q,s) = Size(\mathcal{B}_q^s)/Net(s)$.

Obviously, the query arrival rate to each cloud server is related to the queries assigned to it. For any query q with executing frequency f_q, the cloud server will receive the requests with the rate of f_q, as long as it stores the records queried by q. We consider the query arrival rate to follow the Poisson distribution. Since the additive property of Poisson distribution, the arrival rate of any cloud server s is the summation of all of the execute frequencies of queries on them. Let θ_s denote the mean query arrival rate of server s, such that $\theta_s = \mathcal{F} \cdot \mathcal{A}_{\mathcal{Q}}^s = [f_1, ..., f_M] \cdot [y_{1,s}, ..., y_{M,s}]^\intercal$. We assume that the query processing time of each cloud's database follows the independent exponential distribution. Then we use μ_s to denote the mean query process rate of server s.

Since each query is forwarded to more than one cloud server and processed parallelly, based on the row vector $\mathcal{A}_Q^q$, the final mean response time of each query q equals to the maximum mean response time of all of the execute cloud servers, as follows.

$$T(q,\mathcal{A}) = \max_{y_{q,s} \in \mathcal{A}_Q^q, y_{q,s}=1} \left\{ \frac{\theta_s}{\mu_s - \theta_s} + Tran(q,s) \right\} \tag{8}$$

Thus, for an assignment $\mathcal{A}$ and a sequence of queries $\mathcal{Q} = \{q_1, ..., q_M\}$, the mean response time is shown as follows.

$$T(\mathcal{Q},\mathcal{A}) = \sum_{q \in \mathcal{Q}} T(q,\mathcal{A}) \cdot f_q \tag{9}$$

Definition 3: **Optimal Assignment:** Given a matrix $\mathcal{B}$ built to represent the query result of a sequence of queries $\mathcal{Q}$ on a set of database record $\mathcal{R}$. We use the following optimization problem to find an optimal assignment $\mathcal{A}$ of all of the elements in $\mathcal{B}$ to the cloud servers in $\mathcal{S}$ that minimizes the tradeoff between the total query response time $T(\mathcal{Q},\mathcal{A})$ and information disclosure $D(\mathcal{A})$, where γ is a nonnegative weight and η is the disclosure constraint.

$$\begin{aligned} &\underset{\mathcal{A} \in (\mathcal{B} \times \mathcal{S})}{\text{minimum}} \quad T(\mathcal{Q},\mathcal{A}) + \gamma D(\mathcal{A}) \\ &\text{subject to} \quad D(\mathcal{A}) \leq \eta \end{aligned} \tag{10}$$

IV. ASSIGNMENT OPTIMIZATION

In this section, we describe how to find a record and query assignment with minimal query response time and information disclosure. The optimization problem formulated in equation (10) is NP-hard to solve. We present the following heuristic algorithm to find an approximate optimal assignment.

A. Algorithm Overview

The challenge of solving the optimizing problem (10) lies in the mutual affection between record and query placements. When any record in the result set of a query q is assigned to a cloud server s, then query q also needs to be sent to the same cloud in order to return the entire query result. The response time of query q is reduced, but the workload and information leakage on server s is increased. Hence, instead of determining the placement of record and query separately, we take each element $b_{r,q} \in \mathcal{B}$ as a decision unit. Thus, the optimization problem (10) equals finding an assignment of elements in $\mathcal{B} = \{b_{1,1}, ..., b_{1,M}; ...; b_{N,1}, ..., b_{N,M}\}$ to one of the cloud servers $S_1, ..., S_k$, such that the objective function $G(\cdot)$ is minimized.

Input: matrix $\mathcal{B}$; set of cloud servers $\mathcal{S}$; empty assignment $\mathcal{A}$; objective function $G(\cdot)$
Output: assignment $\mathcal{A}$: $G(\mathcal{A})$ is minimal
Initialization: $\mathcal{A} \leftarrow$ an arbitrary assignment ;
do
  $Stop \leftarrow 0$;
  for *all pairs of cloud servers* $\{s,t\} \in \mathcal{S}$ **do**
    $\mathcal{A}'$ = EXCHANGE$(s,t,\mathcal{A})$;
    if $G(\mathcal{A}') < G(\mathcal{A})$ **then**
      $\mathcal{A} = \mathcal{A}'$;
      $Stop \leftarrow 1$;
    end
  end
while $Stop == 1$;
return $\mathcal{A}$;
Algorithm 1: Optimal assignment search algorithm.

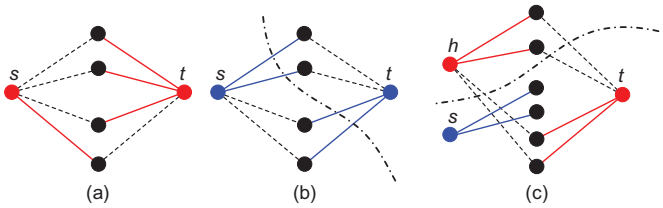


Fig. 2. Search for optimal assignment of elements to a pair of cloud servers via minimum cut.

B. Assignment Minimization via Minimum Cut

The structure of our algorithm is shown in Algorithm 1. Referring to the idea in [9], the algorithm repeatedly executes a for-loop until the first for-loop that cannot make any improvement to the current assignment, the algorithm outputs it. In the for-loop, the algorithm performs the same iteration (EXCHANGE) for each pair of cloud servers. The output of each EXCHANGE is an optimal assignment of input elements to one pair of cloud servers. The variable "Stop" will be set to 1, if the output of any EXCHANGE is an assignment $\mathcal{A}'$ with lower objective function value.

Fig. 2 depicts how to solve EXCHANGE via the minimum $s-t$ cut. We first construct a graph by considering two cloud servers as terminal nodes (s,t) and all the elements in $\mathcal{B}$ as the middle nodes connecting s or t. The solid line in Fig. 2(a)) indicates an assignment of element to a cloud server. In the initial phase of the Algorithm 1, each element in $\mathcal{B}$ is arbitrarily linked to one of the cloud servers in $\mathcal{S}$. In each iteration, the inputs of EXCHANGE are a pair of cloud servers and an assignment $\mathcal{A}$ to be improved. The purpose

of algorithm EXCHANGE is to improve the assignment $\mathcal{A}$ by connecting each element to one of the terminals. The principle of adjustment is to seek a new assignment that with lower objective function value under the constraint of information disclosure. In the initial phase, the algorithm adds edges from both servers to all the elements that are linked to one of the two servers, as shown by the dashed lines in Fig. 2(a)).

Since each element can only link to one of the terminal nodes, this adjustment can be achieved via the minimum $s-t$ cut on Fig. 2(a)). The output of a minimum $s-t$ cut is a set of edges (hit by the dash-dotted lines in Fig. 2(b)) with the minimal cost value to separate terminal node s and t, which is also a better assignment $\mathcal{A}^{'}$ from elements to cloud servers. Some elements previously assigned to server s in $\mathcal{A}$ are now assigned to server t in $\mathcal{A}^{'}$, and vice versa. The assignment of the rest of the cloud servers $h \neq s, t \in \mathcal{S}$ is the same $\mathcal{A}^h = \mathcal{A}^{'h}$. The algorithm repeatedly chooses another pair of cloud servers (s, h), links the elements of s and h with one another, and finds the minimum cut set, as shown in Fig. 2(c).

V. PERFORMANCE EVALUATION

In this section, we conduct experiments to analyze the performance of the assignment strategy output from Algorithm 1. In particular, we demonstrate that our assignment strategy resists access pattern attack, offers high query efficiency and is scalable under different realistic settings.

A. Experimental Settings

1) ***Dataset*** $\mathcal{R}$ ***and queries*** $\mathcal{Q}$*:* We adopt the Enron email dataset for evaluation, which is a set of email documents of 150 Enron corporation employees, sent from 2000 to 2002 [4]. This dataset has 30,109 email documents with 77,000 unique keywords after removing the stopwords [6]. We consider each document as a record, and generate queries by uniformly choosing them from the most frequent keywords. The documents that contain a certain keyword mean the document is the result of the query with that keyword. Since the query frequency does not influence information disclosure, we assume that the frequencies of all of the queries are equal and their sum is 1. So the query arrival rate θ_s of each cloud server is related to the total number of queries assigned to it.

2) ***Cloud Servers*** $\mathcal{S}$*:* Since the service rate of cloud database service is dynamic and difficult to control, we evaluate the performance of our assignment approach via the numerical experiments. We assume that the service rate μ_s of all of the cloud servers is equal to 1 in the following experiments. The time cost of transmitting back each single record is assumed to be 1 ms. Therefore, the bandwidth cost equals to the number of query results on each cloud server. In all of the experiments, we set $\alpha = 0.5$ and $\eta = 0.9$. We use the gco-v3.0 library [10] to implement the minimum cut algorithm and obtain our assignment policies based on the different parameter settings. Table I indicates the specific parameter settings of each figure.

TABLE I
PARAMETER SETTINGS FOR FIGURES

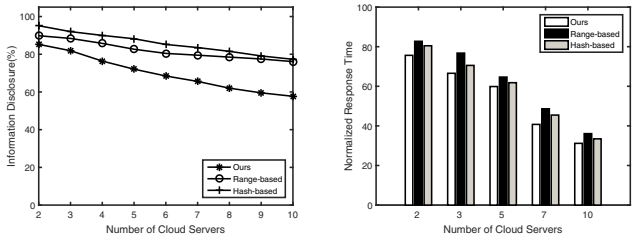
Fig.	No. of $\mathcal{Q}$	No. of $\mathcal{S}$	γ
3(a),3(b)	500	[2,10]	1
4(a),4(b)	[200,1000]	10	1
5(a)	500	10	1
5(b)	500	[2,10]	0, 1

B. Experimental Results

1) ***Benefits of our assignment strategy****:* We compare the record and query assignment strategy designed by our approach with the record placement policies in the distributed databases. The usual sharding policies in the NoSQL database ($e.g.$, MongoDB) are Range-based and Hash-based [11]. In the Range-based sharding, records are divided into different chunks based on the range of shard key values. In the Hash-based sharding, records are evenly distributed among cloud servers based on the hash of shard key values.

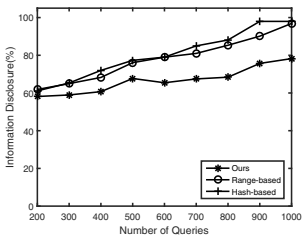
Fig. 3(a) shows the information disclosure of three data assignment policies with the growing number of cloud servers. The leakage always declines with a greater number of servers, since fewer access patterns are observed by each cloud server. Hash-based policy leaks more information than Range-based policy, because the records with "closer" key values are stored on the same server. So queries executed on the same server have similar results. Our assignment strategy discloses the lowest percentage of information over the other two. Under different numbers of cloud servers, our approach saves an average of 14.7% information disclosures compared to the Range-based policy. Compared to the Hash-based policy, our approach can save more. The benefits of our assignment over other policies increases with a greater number of cloud servers. This is because our approach achieves better assignment solutions with more cloud servers.

Fig. 3(b) depicts the response time of three placement strategies. The Hash-based policy has less query response time than the Range-based policy does, because records in Hash-based policy can be parallelly executed on different cloud servers. However, in these two partitioning policies, the query workload on each cloud server has not been considered. In our assignment strategy, using the access pattern as input leads to a faster query response time, which saves an average of 6% of

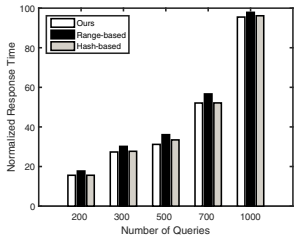


(a) Information disclosure (b) Response time

Fig. 3. Benefits of our assignment strategy on information disclosure and query response time

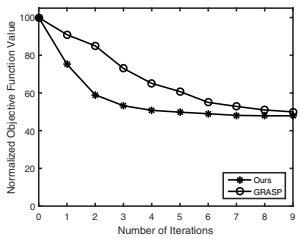


(a) Information disclosure

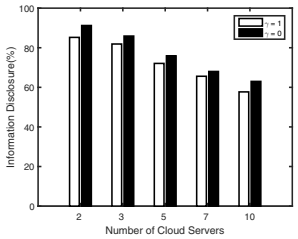


(b) Response time

Fig. 4. Influence of the number of queries on the benefits of our assignment



(a) Convergence speed



(b) Influence of optimization goals

Fig. 5. Evaluation of convergence speed and influence of optimization goals

the query response time compared to the Hash-based policy.

*2) **Influence of the number of queries**:* Fig. 4(a) shows the information disclosure of three data assignment policies with a different number of queries. The leakage increases with a greater number of queries, since more queries are gathered in the same cloud server, and more access patterns are disclosed. Our assignment strategy leaks the lowest percentage of information. Under different numbers of queries, our approach saves an average of 13% information disclosures compared with the Range-based policy. This is because our approach optimizes information disclosures. Fig. 4(b) illustrates the impact of the number of queries on query response time. The response time increase as a greater number of queries. On average, 1.7% response time can be saved by our approach, compared with the Hash-based policy. This result indicates that our approach can avoid information disclosures without sacrificing query efficiency.

*3) **Convergence speed of Algorithm 1**:* Fig. 5(a) demonstrates the comparison of convergence speed between Algorithm 1 (minimum cut) and the greedy randomized adaptive local search procedure (GRASP) proposed in [12]. In each iteration of Algorithm 1, function EXCHANGE is invoked to find an optimal assignment of elements to a pair of cloud servers. In each iteration of GRASP algorithm, function CONSTRUCTION is invoked to greedily find a new assignment with lower objective function value. Fig. 5(a) shows that the objective function value of our algorithm drops quickly in the first several iterations, which indicates that our approach converges faster than GRASP.

*4) **Performance with different optimization goals**:* In the equation (10), γ is an important parameter. When $\gamma = 0$, the optimization goal is only the total query response time under the information disclosure constraint. When the $\gamma = 1$, the optimization goal is both the query response time and information disclosure. Fig. 5(b) compares the information disclosures under different optimization goals by varying the number of cloud servers. For both versions of the optimization problem, the information leakage decreases as the number of cloud servers increases. This result shows that tradeoff optimization goal ($\gamma = 1$) achieves lower information disclosure by using multiple cloud servers.

VI. Conclusion

In this paper, we investigate the problem of access pattern leakage attack to searchable encryption in a cloud database. We adopt multiple cloud deployment by distributing both database records and queries among different cloud servers. To achieve a minimum query response time and information disclosures, we formulate this record and query assignment as an optimization problem and search for the optimal assignment by the minimum $s-t$ cut. The numerical results show that on average 13% access pattern information can be saved by our assignment strategy, without sacrificing query efficiency.

Acknowledgment

This work is supported by the Department of Computing, The Hong Kong Polytechnic University.

References

[1] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[2] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation." in *NDSS*, vol. 20, 2012, p. 12.

[3] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 668–679.

[4] Enron email dataset. [Online]. Available: http://www.cs.cmu.edu/~/enron/

[5] E. Stefanov and E. Shi, "Multi-cloud oblivious storage," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 247–258.

[6] A. Degitz, J. Köhler, and H. Hartenstein, "Access pattern confidentiality-preserving relational databases: Deployment concept and efficiency evaluation." in *EDBT/ICDT Workshops*, 2016.

[7] S. D. C. di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati, "Fragmentation in presence of data dependencies," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 6, pp. 510–523, 2014.

[8] M. A. Abdelraheem, T. Andersson, and C. Gehrmann, "Inference and record-injection attacks on searchable encrypted relational databases," *Cryptology ePrint Archive, Report 2017/024*, 2017.

[9] L. Jiao, J. Lit, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 28–36.

[10] gco-v3.0 library. [Online]. Available: http://vision.csd.uwo.ca/code/gco-v3.0.zip

[11] Mongodb sharding. [Online]. Available: https://docs.mongodb.com/manual/sharding/

[12] T. Rekatsinas, A. Deshpande, and A. Machanavajjhala, "Sparsi: partitioning sensitive data amongst multiple adversaries," *Proceedings of the VLDB Endowment*, vol. 6, no. 13, pp. 1594–1605, 2013.