# Access Pattern Hidden Query over Encrypted Data via Multi-cloud

Yi Dou, Henry C. B. Chan
Department of Computing, The Hong Kong Polytechnic University, Hong Kong
Email: csydou@comp.polyu.edu.hk

***Abstract*—The searchable encryption is one of the solutions supporting the untrusted third party directly searching over the encrypted data. However, recent researches found that it is vulnerable to attacks leveraging the retained statistical property observed from encrypted query results. In this paper, we investigate the problem of the access pattern leakage attack to the searchable encryption. We adopt the multiple cloud deployment by distributing both database records and queries among different cloud servers, so that none of the cloud servers access to the entire database records and queries. To achieve the minimum the 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 up to 32% access pattern information can be saved by our assignment strategy, without sacrificing the response time.**

## I. Introduction

How the cloud providers store and process their consumers' sensitive data become the main concern of the cloud database. Searchable encryption is one of the solutions, which allows directly searching over the encrypted data. The data owner associates each encrypted record with a number of secure indexes. Each index is generated based on one of the plaintext attribute value of the record. In the searching phase, the data owner would transfer the query into a secure trapdoor (encrypted query keywords). Then, the cloud server matches the trapdoor with the indexes to locate the satisfied records.

Most existing searchable encryption schemes are based on the security model defined by Curtmola et al.'s in [1]. However, this model only protects the confidentiality of data at rest, while ignoring the information leakage during the query process. Specifically, the cloud server learns whether the query has been issued before by observing the repetition of trapdoors, which is called as search pattern leakage. The access pattern leakage means that the cloud server learns which encrypted record is included in the result set of which query trapdoor. When the query result sets of any two queries have a large overlap, the cloud server naturally deduce the probability of these two queries keywords appearing together is high. For instance, the keywords 'university' and 'education' often appear together. With the help of probability distribution of the publicly available plaintext datasets, the cloud server is able to reveal the underlying keyword of the trapdoor [2].

Recent research [3] shows that the number of encrypted query results also can be used to reveal the query keywords. In some datasets, more than half of their keywords have unique appearance frequencies [4]. Attackers can reveal the keyword of trapdoor by matching the unique number of results with the frequency of queried keywords in the plaintext dataset.

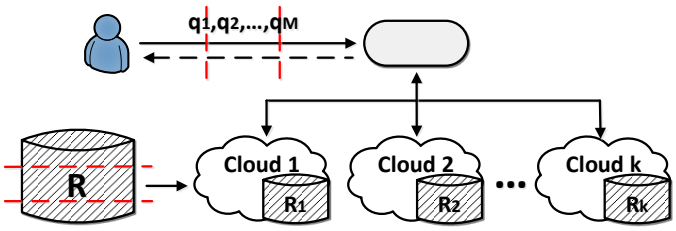



Fig. 1. Assigning records and queries among multiple clouds.

Most previous researches [5] against access pattern leakage attack only discuss in the context of a single server. Similar fragmentation based schemes [6] are proposed to break the sensitive association among attributes and records, which causes the multi-keywords query has to be executed across servers with longer response time. Most of their cost models are based on the assumption that server is alway idle that can immediately process the queries while ignoring the features of the database system and result transmission delay.

To avoid single datacenter outage, many database programs support replication among different datacenters of the same cloud provider [7]. However, the bandwidth between the datacenters in different regions is large enough; even they belong to the same cloud provider. Some cloud providers also have the limitation on the number of replicas per primary database. Hence, deploying database replicas among multiple clouds, but in the same region is a better choice. This deployment has less bandwidth cost and avoids vendor lock-in problem. There is much lower probability for different clouds colluding together to breach the privacy of their users.

To against the access pattern leakage attacks, in this paper, we distribute both database records and queries among the servers of different clouds. As shown in the Fig.1, each cloud server performs independent searchable encryption scheme. Since none of the cloud servers access to the entire database records and queries, it is hard for them to launch the access pattern leakage attack by matching the similar statistical property in the plaintext database. We formulate the record and query assignment as an optimization problem. The objective

is to minimize both the entire query response time and information disclosures. We adopt the minimum $s-t$ cut to search for the optimal assignment. In the experiments, we evaluate the assignment strategy output from our approach with the real world dataset. The numerical results indicate that up to 32% access pattern information can be saved by our assignment strategy, without sacrificing the response time. And our algorithm is scalable under different realistic settings.

The rest of the paper is organized as follows. Section II presents the security problem and strategies. Section III formulates our proposed record and query assignment model. Section IV describes how to search the optimal assignment strategy by solving the assignment optimization. The experiment results and conclusion are shown in Section V and VI.

## II. SECURITY STRATEGIES

In this section, we are going to 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) proposed an attack to recover the keyword of trapdoor in the SSE (searchable symmetric encryption) schemes [2]. It makes use of the unique co-occurrence probability among keywords. The attacker has two matrixes $M_p$ and $M_c$, $M_p$ is obtained from the public plaintext datasets before the searching, $M_c$ is built by observing the trapdoors $T_q$ (encrypted keywords), and their corresponding query results $R_q$ (encrypted records) during the SSE searching. Hence, the process of revealing the keywords of observed trapdoors is to find matching elements between these two matrixes. Each matrix element is a co-occurrence probability of any two keywords or trapdoors appears 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 the $i$th and $j$the trapdoors in $M_c$ to the plaintext keywords of the $u$th and $v$the queries in $M_p$.

*2) Count Attack:* After IKK attack, Cash et al. [3] proposed another efficient attack to recover the keyword of a trapdoor. Since some keywords have the unique frequency in the dataset, the attacker exploits this property to map the trapdoors to their plaintext keywords as long as the attacker have the entire encrypted records. Firstly, the attacker calculates the frequency of each keyword $w$ in the publicly plaintext dataset, that is $count(w)$. Secondly, the server counts the number of records in each trapdoor query result during the SSE searching, which is also the frequency of trapdoor keyword in the encrypted dataset, that is $count(R_q)$. When there are any keywords in the plaintext dataset has the same unique frequency as the trapdoor in SSE, that is $count(w) = count(R_q)$, the attacker can directly associate the keyword $w$ with the trapdoor $T_q$.

### B. Security Strategies

Both IKK and count attack highly relies on the accuracy of background knowledge extracted from plaintext dataset [8]. The successful recovery rate would decline quickly, if the attacker only obtains a fraction of plaintext dataset [3]. Conversely, when the server only observes partial information on the encrypted dataset, it also can prevent the above attacks, even with the complete knowledge in 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 trapdoors, the attacker is hard to deduce their keywords by matching their frequencies. Padding is one of the approaches to hide actual query result by adding dummy records and identifiers before building the secure indexes [2]. However, padding unsatisfied records cause the false positive in the query results and bring extra communication overhead [3], [8]. In this paper, we adopt the multiple cloud deployment. We are going to remove the records in the trapdoor query results to other cloud servers, so that to make the co-occurrence probability and keyword frequency observed by the single server is incorrect. In other words, each server can only return part of the correct query results. The same query need 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 the padding approach, we are going to distribute queries with similar results on the same server, so that to make the query result of trapdoors on the same cloud server to be as similar as possible.

## III. ASSIGNMENT STRATEGY

The multiple cloud deployment shown in Fig.1 often follows the problem about "which" record and query should be placed on "which" cloud server. In this section, we are going to introduce the strategy of record and query assignment, so that to satisfy above introduced the 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 each of which 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 are going to be assigned both records and query tasks. Before distribution, we build a matrix $\mathcal{B}$ to describe the query results of all 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 $q$th column vector of $\mathcal{B}$ denote the query result of $q$, that is $\mathcal{B}_q$.

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

### 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 single cloud server cannot reveal the statistical property of the original access pattern.

*Definition 1:* **Record and Query Assignment:** We define $\mathcal{A}$ as an assignment of records $\mathcal{R}$ and queries $\mathcal{Q}$ to a set of cloud servers $\mathcal{S}$, which is a process of horizontally and vertically partitioning 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 the repetition of the returned results, each element $b_{r,q} \in \mathcal{B}$ will be assigned to one of the cloud servers.

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

To formulate the cost of an assignment $\mathcal{A}$, we use matrix $\mathcal{A}_R$ and $\mathcal{A}_Q$ to describe the placement of records and queries on cloud servers, respectively. 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 no zero row, that is $\sum_{s=1}^{s=k} x_{r,s} \geq 1$.

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

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 has to be distributed to at least one cloud server, $\mathcal{A}_Q$ has no 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 $s$th 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 together. 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 are going to 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 together after applying different searchable encryption schemes. We firstly analyse the information disclosure on each single cloud server. By observing the response results of all 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 $q$th 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 is unsatisfied 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 query on 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 all 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 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 $f_q$, 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 queries arrival rate follows the poisson distribution. Since the additive property of poisson distribution, the arrival rate of any cloud server $s$ is the summation of all the execute frequencies of queries on them. Let $\theta_s$ 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 $\mu_s$ 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 $\mathcal{A}_{\mathcal{Q}}^q$, the final mean response time of each query $q$ equals to the maximum mean response time of all the execute cloud servers, as follows.

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

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.

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

*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 the elements in $\mathcal{B}$ to the cloud servers in $\mathcal{S}$ that minimizes both the total query response time $T(\mathcal{Q}, \mathcal{A})$ and information disclosure $D(\mathcal{A})$.

$$G(\mathcal{A}) = \min_{\mathcal{A} \in (\mathcal{B} \times \mathcal{S})} T(\mathcal{Q}, \mathcal{A}) + D(\mathcal{A}) \quad (10)$$

## IV. Assignment Optimization

In this section, we describe how to find a record and query assignment with the 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 need 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 to find 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**
  $Stop \leftarrow 0$;
  **forall** *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.

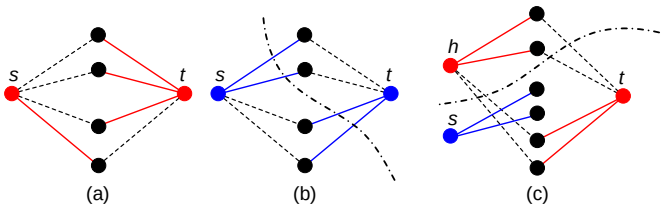



Fig. 2. Search for optimal assignment of elements to a pair of cloud servers via minimum cut.

### B. Assignment Minimization via Minimum Cut

The structure of our algorithm is shown in Algorithm 1. Referring to the idea in [9], [10], the algorithm repeatedly executes the for-loop until the first for-loop that doesn't change the variable "Stop" to 1. 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. Once any for-loop cannot make any adjustment to the current assignment $\mathcal{A}$, the algorithm outputs $\mathcal{A}$.

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 $s$ and $t$. In the initial phase of the Algorithm 1, each element in $\mathcal{B}$ is arbitrarily connected to one of the cloud servers in $\mathcal{S}$. Given the input of a cloud server pair and an assignment $\mathcal{A}$ to be updated, the algorithm adds edges from both servers to all the elements that was linked to one of the two servers, as shown in Fig.2(a)). The weight of the edge between an element and a terminal node represents the cost of assigning an element to cloud server $s$ or $t$, which includes the additional information leakage and the response time change of all the queries placed on that server (caused by increased arrival rate and result size).

Therefore, the output of a minimum $s-t$ cut on the above constructed graph is an edge set (hit by the dashed lines in Fig.2(b)) with the minimal cost value to separate terminal node $s$ and $t$, which is also a new 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 cloud servers $h \neq s,t \in \mathcal{S}$ are the same in the two assignments $\mathcal{A}^{h} = \mathcal{A}^{'h}$. The algorithm repeatedly chooses another pair of cloud servers $(s,h)$, links the elements of $s$ and $h$ with each others, and finds the minimum cut set, as shown in Fig.2(c).

## V. PERFORMANCE EVALUATION

In this section, we conduct experiments to analyse 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 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 during 2000 to 2002 [4]. This dataset has 30,109 email documents with 77,000 unique keywords after removing the stopwords [5]. 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 means the document is the result of the query with that keyword. Since the query frequency does not influence the information disclosure, we assume the frequencies of all the queries are equal and their sum is 1. So the query arrival rate $\theta_s$ of each cloud server is related to the total number of queries assigned to it.

| Fig. | No. of $\mathcal{Q}$ | No. of $\mathcal{S}$ | $\alpha$ |
|---|---|---|---|
| 3(a),3(b) | 500 | [1,10] | 0.5 |
| 4(a),4(b) | [100,1000] | 10 | 0.5 |
| 5(a) | 500 | 10 | 0.5 |
| 5(b) | 500 | 5, 10 | [0,1] |

TABLE I
PARAMETER SETTINGS FOR FIGURES

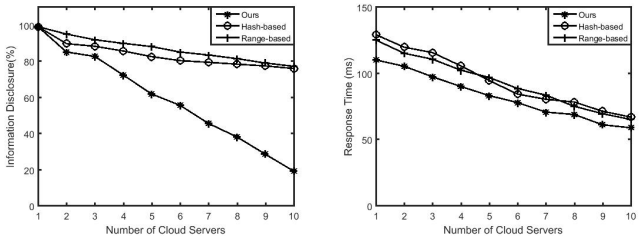
*2)* ***Cloud Servers $\mathcal{S}$:*** Since the service rate of cloud database service is dynamic and hard to control, we evaluate the performance of our assignment approach via the numerical experiments. We assume that the service rate $\mu_s$ of all the cloud servers are equal to 1 in the following experiments. The time cost of transmitting back each single record is assumed to be 1 ms. So the bandwidth cost equals to the number of query result on each cloud server. We use the gco-v3.0 library [11] to implement the minimum cut algorithm and obtains our assignment policies based on the different parameter settings. Table I indicates the specific parameter settings of each figure.

### B. Experimental Results

*1)* ***What are the benefits of our assignment?:*** We compare the record and query assignment strategy designed by our approach with the record placement policies in the distributed databases. The usual placement policies in the NoSQL database ($e.g.$, MongoDB) are Range-based and Hash-based. In the Range-based policy, records are divided into different chunks based on the shard key values. Each chunk is then placed at one of the cloud servers. Records with "closer" shard key values are stored on the same server. In the Hash-based policy, records with "closer" shard key values are placed on the different servers. It distributes the records evenly among cloud server by computing the hash of shard key values.

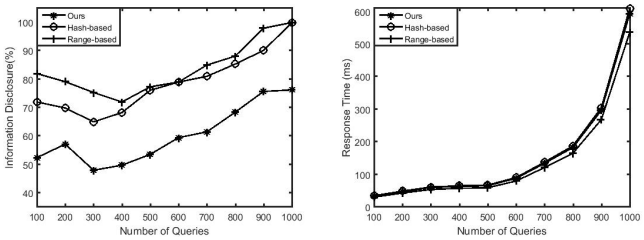
Fig.3(a) shows the information disclosure of three data assignment policies with the growing number of cloud servers. The leakage always declines as a greater number of servers, since fewer access patterns are observed by each cloud server. Range-based policy leaks more information than Hash-based, because the records assigned based on hash value can disassociate the connection between records. Our assignment strategy discloses the lowest percentage of information over other two. Compared with Hash-based policy, our approach saves 10.67% disclosures on the queries with insecure hamming distances and 21.3% of the insecure queries with original frequency. Compared with Range-based policy, our approach can save more disclosures. The benefits of our assignment over other policies increases with the 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 Range-based policy has less query response time



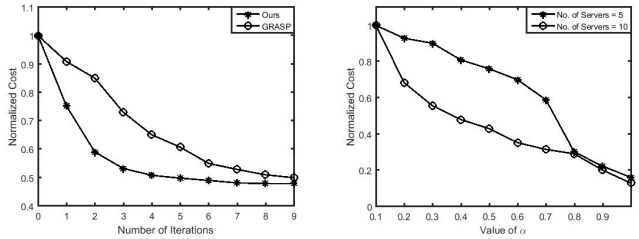

(a) Information disclosure (b) Response time

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




(a) Information disclosure (b) Response time

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




(a) Converge speed (b) Influence of parameter $\alpha$

Fig. 5. Evaluation of converge speed and influence of parameter $\alpha$

than Hash-based, because records of the same query are scattered among different cloud servers. However, in these two partitioning policies, the amount of queries on each cloud server has not been considered. In our assignment strategy, using the access pattern as input leads to faster query response time which saves 12% response time than other two polices.

*2) **How does the number of queries influence the benefits?**:* Fig.4(a) shows the information disclosure of three data assignment policies with the different number of queries. The access pattern leakage of Range-based and Hash-based policy increases when the number of queries is too small and too large, since when more queries are gathered in the same cloud server and more access patterns are disclosed. Compared with Hash-based policy, our approach saves 23.69% information disclosures on the insecure queries. This is because our approach optimizes the information disclosure. Fig.4(b) illustrates the impact of the number of queries on the query response time. Compared with the other two policies, our approach has almost the same query efficiency. This result indicates that our approach optimizes the information disclosures than the query response time.

*3) **How fast does Algorithm 1 converge?**:* Fig.5(a) demonstrates the comparison of converge speed between Algorithm 1 (minimum cut) and greedy randomized adaptive local search procedure (GRASP) proposed in [12]. In the each iteration of our algorithm, function EXCHANGE (minimum $s-t$ cut) is invoked once to find an optimal assignment of elements to a pair of cloud servers. In the each iteration of GRASP algorithm, function CONSTRUCTION is invoked once to greedily find a new assignment with lower objective function value. Fig.5(a) shows the cost of our algorithm drops quickly in the first several iterations, which indicates that our approach converges faster than the GRSAP algorithm. In the Fig.5(b), we evaluate the cost to solve the optimal assignment by varying the security threshold value $\alpha$ from 0.1 to 1. The cost decreases with $\alpha$ approaches to 1, since fewer number of queries are inserted into the set of insecure queries.

## VI. Conclusion

This paper investigated the problem of the access pattern leakage attack to the searchable encryption in the cloud database. We adopt the multiple cloud deployment by distributing both database records and queries among different cloud servers. To achieve the minimum the 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 up to 32% access pattern information can be saved by our assignment strategy, without sacrificing the response time.

## Acknowledgment

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

[6] 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.

[7] Mongodb multi-data center deployments. [Online]. Available: http://s3.amazonaws.com/info-mongodb-com/MongoDB_Multi_Data_Center.pdf

[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] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

[10] 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.

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

[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.