Light-weight Preservation of Access Pattern Privacy in Un-trusted Storage

Ka Yang[1], Jinsheng Zhang[2], Wensheng Zhang[2], and Daji Qiao[1]

[1] Electrical and Computer Engineering, Iowa State University / Ames, IA 50010, USA {yangka, daji}@iastate.edu
[2] Computer Science, Iowa State University / Ames, IA 50010, USA {alexzjs, wzhang}@iastate.edu

***** Corresponding Author: Ka Yang

Received July 14, 2013; Revised July 29, 2013; Accepted August 12, 2013; Published October 31, 2013

** Extended from a Conference: Preliminary results of this paper were presented at the 16th European Symposium on Research in Computer Security (ESORICS'11).This present paper has been accepted by the editorial board through the regular reviewing process that confirms the original contribution.*

** Regular Paper*

***Abstract*:** With the emergence of cloud computing, more and more sensitive user data are outsourced to remote storage servers. The privacy of users' access pattern to the data should be protected to prevent un-trusted storage servers from inferring users' private information or launching stealthy attacks. Meanwhile, the privacy protection schemes should be efficient as cloud users often use thin client devices to access the data. In this paper, we propose a lightweight scheme to protect the privacy of data access pattern. Comparing with existing state-of-the-art solutions, our scheme incurs less communication and computational overhead, requires significantly less storage space at the user side, while consuming similar storage space at the server. Rigorous proofs and extensive evaluations have been conducted to show that the proposed scheme can hide the data access pattern effectively in the long run after a reasonable number of accesses have been made.

***Keywords*:** Privacy, Access pattern, Un-trusted storage, Cloud computing

1. Introduction

With the emergence of cloud computing, data outsourcing is becoming more and more prevalent. Cloud storage [1, 2] (such as Amazon Simple Storage Service and Rackspace Cloud Files) enables enterprise and individual users to enjoy flexible, on-demand and high-quality services, without the need to invest on expensive infrastructure, platform or maintenance. Although envisioned as a promising service platform for the next-generation of Internet, cloud computing is facing great privacy and security challenges that may impede its fast growth and increased adoption if not well addressed. For example, one of the security concerns for the user is that the service provider itself may breach the users secrecy and privacy. For example, Google has been criticized for tapping into its users data. Rising to the challenges, researchers have proposed many schemes [3-5] to protect the confidentiality and integrity of users' outsourced data. Unfortunately, limited research has been conducted on the protection of users' privacy during their access to the storage server, such as the access frequency to each data item and the linkage between accesses of data items. Leakage of such access pattern information may enable potential privacy attacks such as focused attacks against selected data items. Servers may also infer a user's activity pattern or private interest by tracking the user's access to a particular data item.

To strictly protect the privacy of data access pattern, the intention of every data access operation should be hidden so that observers of the operations cannot gain any meaningful information. Conforming to this strict requirement of access pattern privacy, Chor *et al.* [6], Ostrovsky *et al.* [7] and Itkis [8] introduced the notions of the private information retrieval (PIR) in an information theoretical setting and the computational PIR by restricting the database to perform only polynomial-time computations. Fully implementing the PIR notion is, however, expensive. As shown by Sion *et al.* [9], deployment of any single-server PIR protocol is not necessarily more efficient than a simple transfer of the entire

database. Another approach to the strict preservation of data access pattern privacy is based on the notion of oblivious RAM (ORAM) [10-19]. In a latest ORAM implementation [16], about log n data items of the database should be scrambled every time after a single data item has been requested, where n is the total number of data items in the database. Let τ denote the size of a data item in bits. This ORAM scheme requires at least $O(\sqrt{n} \cdot \tau)$user storage and incurs a communication and computational complexity of $O(\log n \cdot \tau)$. The cost of this scheme is still rather expensive especially when the number of data items is large and the data are accessed frequently.

Although strict protection of data access pattern privacy is attractive, less strict protection, such as protecting the privacy of long-term access pattern, is also very useful in practice. For example, a malicious server may use the statistical data access pattern of a user to infer the user's private information or conduct stealthy attacks. Moreover, being lightweight is also highly desired by users, as many of them often access the cloud with thin client devices such as smartphones. Based on these considerations, we propose a lightweight scheme to preserve the privacy of long-term data access pattern in this paper. The outline of the proposed scheme is as follows. Every time when a data item is needed by a user, (i) the user retrieves the desired data item together with additional dummy data items to hide the actual retrieval target; and (ii) the retrieved data items are re-encrypted and re-positioned before being stored back to the server to perturb the connections between data items and their storage locations at the server. The scheme records the storage locations of data items in index files, which are stored in a pyramid-like hierarchical structure at the storage server to reduce communication, computational and storage overheads. Similar to data items, the access pattern to index files is also protected with additional dummies and re-positioning of the files after access. A set of delicately designed rules are used in the selection of dummy data items and index files as well as the repositioning of the files, which ensures that the connections between data items and their storage locations are reshuffled gradually, become more and more difficult to trace as the number of accesses increases, and eventually become fully untraceable. Rigorous proofs and extensive evaluations have been conducted to demonstrate that the proposed scheme can hide the data access pattern in the long run, and the number of accesses required to preserve the access pattern privacy is reasonable in many situations.

The rest of the paper is organized as follows. Section 2 describes the system models. The proposed scheme is elaborated in Section 3. Section 4 and Section 5 analyze its security performances and convergence rate respectively. Section 6 reports the evaluation results and Section 7 presents the overhead analysis. Section 8 discusses the related work. Finally, Section 9 concludes the paper.

2. Models and Assumptions

2.1 System Model

We consider a basic remote storage system with a storage server and a single user. The user stores its sensitive data on the remote server, which in turn provides an online interface for the user to access the outsourced data. Later on, when the need for a data item arises, the user requests it from the server, updates the data item after usage, and then uploads the updated data item back to the server. Similar to [10-19], we assume that all the data items stored at the server have the same size so the server cannot identify a data item from its size. In practice, this can be achieved conveniently by appending padding bits to short data items or dividing large data items into smaller ones.

2.2 Security Model

We assume that the storage server is curious about the user's private information and may launch malicious attacks. Specifically, it may be interested in obtaining the user's data access pattern over the long term, which primarily includes the following information: *which data items that have been requested by the user and the number of times that a particular data item has been requested by the user.*

If the access pattern information is obtained, the server may be able to launch various attacks. For example, the server may attempt to infer the user's activity pattern or private interest via tracking the user's access to some particular data items. The server may also launch focused attacks towards user's data that are accessed with very high frequency, or stealthily delete data that are never accessed to save its storage and maintenance costs without being noticed by the user.

As for the user, we assume that it has a primitive encryption function that generates different cipher-texts over different input, and the server does not have non-negligible advantage at determining whether a pair of encrypted items of the same length represent the same data item. We assume that data confidentiality and integrity are protected using existing techniques and the communication channel between the user and the server is secured using mechanisms such as SSL/IPSec. We do not consider denial of service attacks or timing attacks as they can be addressed independently from this work.

2.3 Design Goal

Our main design goal is to develop a lightweight solution to prevent the server from knowing the user's long-term access pattern to the data stored at the server, while allowing the user to access the outsourced data with low communication and computational overhead. Specifically, we preserve the access pattern privacy by breaking the connections between the data items and their storage locations gradually.

3. The Proposed Scheme

3.1 System Setup

Before describing our proposed scheme in detail, we first explain the system setup.

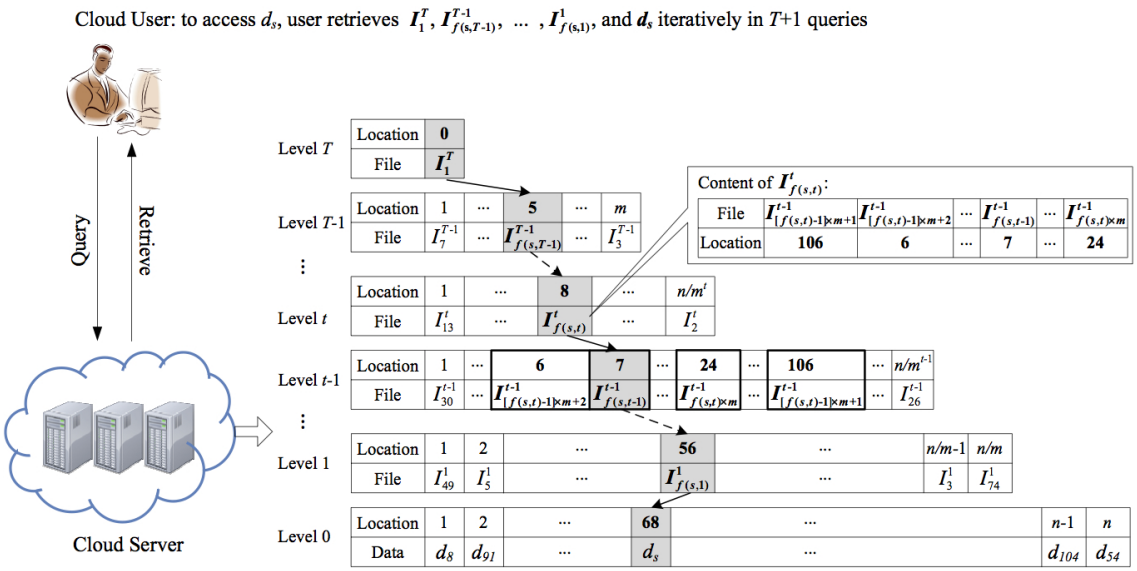


Fig. 1. System setup. Data items and index files form a pyramid-like hierarchical storage structure at the server. Each index file records the storage locations of *m* index files at its next lower level. For example, the content of $I_{f(s,t)}^t$ is shownin the callout box, and the *m* level-(*t* − 1) index files associated with $I_{f(s,t)}^t$ are shown as bold boxes in the figure. Here, $f(s,t)=\left\lceil \frac{s}{m^t} \right\rceil$. To obtain data item d_s, the user performs a sequence of queries iteratively in a top-down manner, to obtain *T* index files (marked as gray boxes), one at each level of the hierarchy.

3.1.1 Hierarchical Storage Structure at the Server

We study a system where a user stores n distinct data items (denoted by d_i, $i = 1, \cdots, n$) at a storage server. All data items are encrypted with the user's secret key before uploading. In addition to data items, the server stores a hierarchy of index files with the following features:

- As shown in Fig. 1, there is a total of $T = \lceil \log_m n \rceil \geq 1$ levels of index files, where $m > 1$ is a design parameter. In Section 7, we analyze the relation between m and the communication, computational and storage overheads incurred by our solution. To simplify the presentation, we assume that $\log_m n$ is an integer in the rest of the paper.
- At level t ($t = 1, \cdots, T$), there are $\frac{n}{m^t}$ index files(denoted by I_j^t, $j = 1, \cdots, \frac{n}{m^t}$). So the total number of index files in the hierarchy is $\sum_{t=1}^{T} \frac{n}{m^t} = \frac{n-1}{m-1}$.
- Each index file records the storage locations of m index files at its next lower level. Specifically, I_j^t at level t contains the storage location information of the following index files at level (t − 1): $I_{(j-1)m+1}^{t-1}$, $I_{(j-1)m+2}^{t-1}$, $\cdots$, I_{jm}^{t-1}, as illustrated in the callout box in Fig. 1.
- There is only a single index file at the top level (i.e., level T): I_1^T.
- Data items form the bottom level (i.e., level 0) of the hierarchy.
- Files at different levels of the hierarchy are stored at non-overlapping storage spaces.

Note that, as shown in Fig. 1, there is no fixed order-correspondence between an index file (or a data item) and its storage location. This is due to the design nature of our proposed scheme, whose key idea is to randomize the storage locations of index files and data items after each access. Details will be discussed in Section 3.2.

3.1.2 Iterative Query Process by the User

With such a pyramid-like hierarchical storage structure, we have the following observation about the relation between a data item and its index files: *the storage location of the data item d_s is recorded in the level-1 index file $I_{f(s,1)}^1$, whose storage location information is in turn recorded in the level-2 index file $I_{f(s,2)}^2$, so on and so forth, till the top-level index file I_1^T; here, f(s,t) is defined as f(s,t)* $= \left\lceil \frac{s}{m^t} \right\rceil$. This relation is illustrated in Fig. 1 as a linked chain of gray boxes from top level T to bottom level 0.

Based on the above observation, we know that the user can obtain the desired data item d_s by performing a sequence of queries to obtain these T index files in the chain: $I_1^T, I_{f(s,T-1)}^{T-1}, \cdots, I_{f(s,1)}^1$ in a top-down manner

through the hierarchy; once $I^1_{f(s,1)}$ is obtained, the user gets to know the storage location of d_s and can then issue the final query to obtain the data item. After the access, the data items and index files are updated, re-encrypted and uploaded back to the server.

To simplify the presentation, we assume that the user requests the data items in rounds and the user requests a single data item in each round. In the following, we explain our proposed scheme in detail. Table 1 lists the notations to be used in the rest of the paper.

Table 1. Notations Used in the Paper

Notation	Description
n	the total number of data items
D	the set of all data item IDs
m	the number of storage locations recorded in an index file
I^t_j	the j-th index file at level t of the hierarchy
$\xi(j,t)$	the set of IDs of files whose storage locations are recorded in the level-t index file of ID j
$\mathcal{L}^t$	the set of storage locations of level-t files
$f(i,t)$	the ID of the index file that corresponds to data item d_i at level t
$\mathcal{Q}^t_{pre}$ $(t \geq 1)$	the set of IDs and locations of level-t index files accessed in the previous round
$\mathcal{Q}^t_{cur}$ $(t \geq 1)$	the set of IDs and locations of level-t index files accessed in the current round
$\mathcal{Q}^0_{pre}$	the set of IDs and locations of data items accessed in the previous round
$\mathcal{Q}^0_{cur}$	the set of IDs and locations of data items accessed in the current round

3.2 Scheme Description

3.2.1 Scheme Overview

Our proposed scheme is executed every time when the user needs to request a data item. The key ideas of the scheme include: (i) extra dummy data items and index files (called dummies for short) are requested to hide the actual files of the user's interest; (ii) multiple dummies are selected so that the user's request at each round has the same format, which is a necessity to hide the access pattern [10] and (iii) the retrieved files are re-encrypted and re-positioned before being stored back to the server so as to break the connections between files and their storage locations at the server. Generally, these rules ensure that the connections between files and their storage locations are reshuffled gradually, become more and more difficult to trace as the number of accesses increases, and eventually become fully untraceable. Detailed explanations and analysis will be presented in the following sections.

Assumption: The following assumption is made on the initial condition when our scheme starts: *for any* $t = 1, \cdots, T-1$, *the mappings between level-*t *and level-*$(t-1)$ *files are unknown to the server*. In other words, for any particular data item, the server has no knowledge about the corresponding index files; similarly, for any particular index file, the server has no knowledge about the corresponding index files at the upper layers.

Algorithm 1 Proposed Access Procedure of a User

Step 1: Selection of Data Items (of IDs $\mathcal{Q}^0_{\text{cur}}.D_R$ and $\mathcal{Q}^0_{\text{cur}}.D_S$) to Access
1: $\mathcal{Q}^0_{\text{cur}}.D_R \leftarrow UserRequest()$; $k \leftarrow UserKey()$; // input user's desired data and secret key
2: $Download\&Decrypt_k(\mathcal{Q}^0_{\text{pre}}, Hist[0])$: // get access history of data items from location $Hist[0]$ & decrypt it
3: **if** $\mathcal{Q}^0_{\text{cur}}.D_R \in \{\mathcal{Q}^0_{\text{pre}}.D_R, \mathcal{Q}^0_{\text{pre}}.D_S\}$ **then**
4: $\quad \mathcal{Q}^0_{\text{cur}}.D_S \leftarrow RandomSelectOne(\mathcal{D} \setminus \{\mathcal{Q}^0_{\text{cur}}.D_R\})$;
5: **else**
6: $\quad \mathcal{Q}^0_{\text{cur}}.D_S \leftarrow RandomSelectOne(\{\mathcal{Q}^0_{\text{pre}}.D_R, \mathcal{Q}^0_{\text{pre}}.D_S\})$;
7: **end if**
Step 2: Query for Index Files and Data Items
1: $Download\&Decrypt_k(I^T_1, 0)$; // download top-level index file from location 0 & decrypt it
2: $\mathcal{Q}^T_{\text{cur}}.D_R \leftarrow 1$; $\mathcal{Q}^T_{\text{cur}}.D_S \leftarrow 1$; $\mathcal{Q}^T_{\text{cur}}.D_N \leftarrow 1$;
3: **for** $(t \leftarrow (T-1); t \geqslant 0; t--)$ **do**
4: $\quad$ // Step 2.1: Selection of Level-t Index Files and $\mathcal{Q}^0_{cur}.L_N$
5: $\quad$ **if** $t > 0$ **then**
6: $\quad\quad Download\&Decrypt_k(\mathcal{Q}^t_{\text{pre}}, Hist[t])$; // get access history of level-t index files
7: $\quad\quad \mathcal{Q}^t_{\text{cur}}.D_R \leftarrow f(\mathcal{Q}^0_{\text{cur}}.D_R, t)$; $\mathcal{Q}^t_{\text{cur}}.D_S \leftarrow f(\mathcal{Q}^0_{\text{cur}}.D_S, t)$; // find out files storing level-t indices of data items $\mathcal{Q}^0_{cur}.D_R$ and $\mathcal{Q}^0_{cur}.D_s$
8: $\quad\quad$ **if** $\mathcal{Q}^t_{\text{cur}}.D_R = \mathcal{Q}^t_{\text{cur}}.D_S$ **then**
9: $\quad\quad\quad \mathcal{Q}^t_{\text{cur}}.D_S \leftarrow RandomSelectOne(\xi(\mathcal{Q}^{t+1}_{\text{cur}}.D_R, t+1) \cup \xi(\mathcal{Q}^{t+1}_{\text{cur}}.D_S, t+1) \setminus \{\mathcal{Q}^t_{\text{cur}}.D_R\})$;
10: $\quad\quad$ **end if**
11: $\quad$ **end if**
12: $\quad$ **if** $\{\mathcal{Q}^t_{\text{cur}}.D_R, \mathcal{Q}^t_{\text{cur}}.D_S\} \subseteq \mathcal{Q}^t_{\text{pre}}$ **then**
13: $\quad\quad \mathcal{Q}^t_{\text{cur}}.L_N \leftarrow RandomSelectOne(\mathcal{L}^t \setminus \{\mathcal{Q}^t_{\text{pre}}.L_R, \mathcal{Q}^t_{\text{pre}}.L_S, \mathcal{Q}^t_{\text{pre}}.L_N\})$;
14: $\quad$ **else**
15: $\quad\quad$ **if** $\mathcal{Q}^t_{\text{cur}}.D_R \in \mathcal{Q}^t_{\text{pre}}$ **then**
16: $\quad\quad\quad \mathcal{Q}^t_{\text{cur}}.L_N \leftarrow RandomSelectOne(\{\mathcal{Q}^t_{\text{pre}}.L_R, \mathcal{Q}^t_{\text{pre}}.L_S, \mathcal{Q}^t_{\text{pre}}.L_N\} \setminus \{\mathcal{Q}^t_{\text{cur}}.L_R\})$;
17: $\quad\quad$ **else**
18: $\quad\quad\quad \mathcal{Q}^t_{\text{cur}}.L_N \leftarrow RandomSelectOne(\{\mathcal{Q}^t_{\text{pre}}.L_R, \mathcal{Q}^t_{\text{pre}}.L_S, \mathcal{Q}^t_{\text{pre}}.L_N\} \setminus \{\mathcal{Q}^t_{\text{cur}}.L_S\})$;
19: $\quad\quad$ **end if**
20: $\quad$ **end if**
21: $\quad Download\&Decrypt_k(\mathcal{Q}^t_{\text{cur}}.\{D_R, D_S, D_N\}, \mathcal{Q}^t_{\text{cur}}.\{L_R, L_S, L_N\})$; /* download files of IDs $\mathcal{Q}^t_{cur}.\{D_R, D_S, D_N\}$ from locations specified by $\mathcal{Q}^t_{cur}.\{L_R, L_S, L_N\}$ respectively but in an arbitrary order & decrypt them */
$\quad$ // Step 2.2: Random Reshuffling
22: $\quad$ **if** $RandomSelectOne(\{0,1\}) = 1$ **then**
23: $\quad\quad Swap(\mathcal{Q}^t_{\text{cur}}.D_R, \mathcal{Q}^t_{\text{cur}}.D_S)$;
24: $\quad\quad$ Update $\mathcal{Q}^t_{\text{cur}}.\{L_R, L_S\}$ in level-$(t+1)$ index files of IDs $\mathcal{Q}^{t+1}_{\text{cur}}.\{D_R, D_S\}$;
25: $\quad$ **end if**
$\quad$ // Step 2.3: Reencryption/Uploading of Level-$(t+1)$ Files and Level-t Access History
26: $\quad Reencrypt_k\&Upload(\mathcal{Q}^{t+1}_{\text{cur}}.\{D_R, D_S, D_N\}, \mathcal{Q}^{t+1}_{\text{cur}}.\{L_R, L_S, L_N\})$; /* reencrypt & upload files of IDs $\mathcal{Q}^{t+1}_{cur}.\{D_R, D_S, D_N\}$ to locations specified by $\mathcal{Q}^{t+1}_{cur}.\{L_R, L_S, L_N\}$ respectively but in an arbitrary order */
27: $\quad Reencrypt_k\&Upload(\mathcal{Q}^t_{\text{cur}}, Hist[t])$;
28: **end for**
Step 3: Re-encryption and Uploading of Accessed Data Items
1: $Reencrypt_k\&Upload(\mathcal{Q}^0_{\text{cur}}.\{D_R, D_S, D_N\}, \mathcal{Q}^0_{\text{cur}}.\{L_R, L_S, L_N\})$;

Fig. 2. Pseudo-code of the proposed scheme.

Data Structures Recording Access History: Our scheme makes use of past file access history when selecting dummies. To facilitate such mechanism, the historical information about the previous round of file access at layer t is recorded in a data structure denoted as $\mathcal{Q}^t_{pre}$, which consists of six fields: D_R, D_S and D_N recording the file IDs, and L_R, L_S and L_N recording their storage locations, respectively. The data structures are stored in cipher-text in a designated storage space at the server, and we denote the storage location of $\mathcal{Q}^t_{pre}$ as $Hist[t]$.

Structure of the Algorithm: The pseudo-code of our scheme is presented in Fig. 2. The scheme starts by selecting dummy data items. Then, it works iteratively to select, download, process and upload the index files, from the top level to the bottom level of the index hierarchy. In

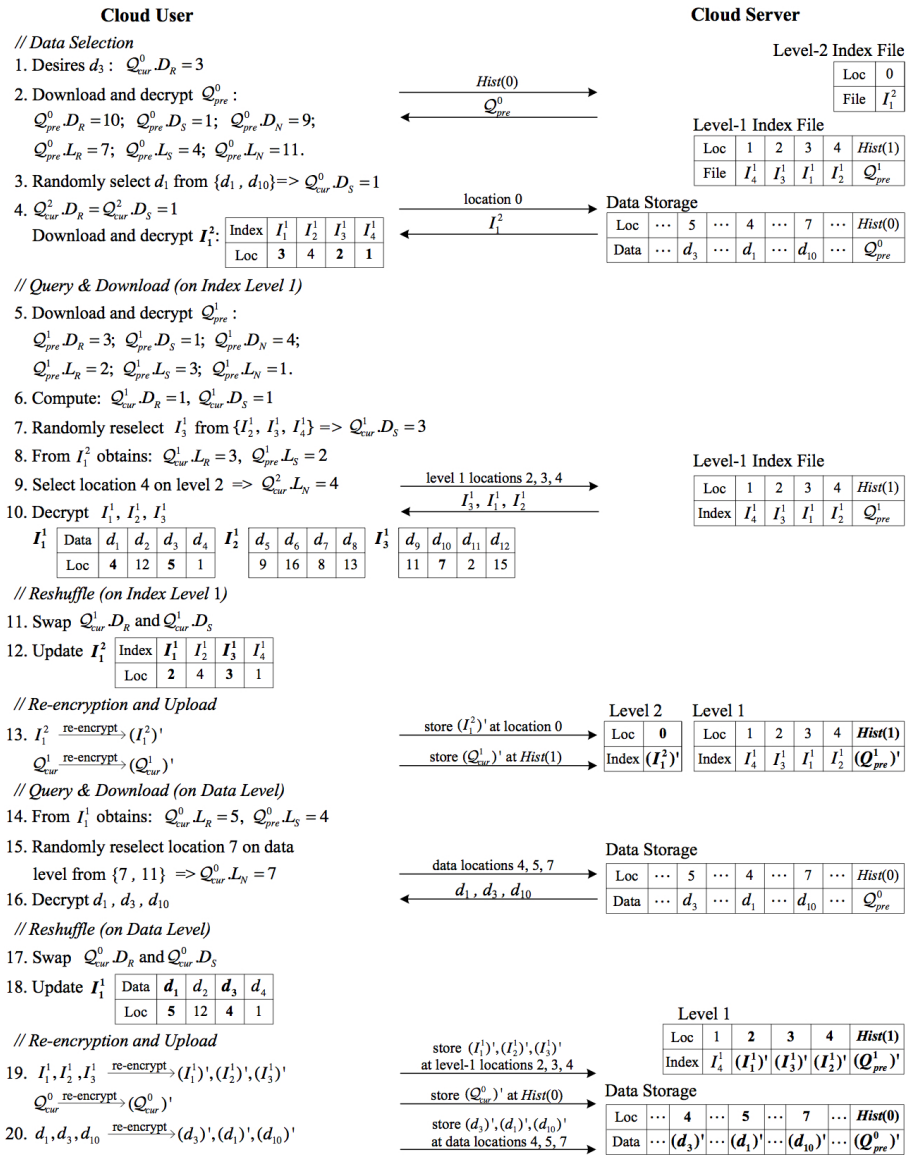


Fig. 3. An example of the access procedure of a user. There is a total of n = 16 data items and T = 2 levels of index files stored at the server. We use d_i' to represent that data item d_i appears differently after re-encryption. In this example, data items d_1, d_9, d_{10} were accessed in the previous round. It shows how the user operates when it is interested in obtaining data item d_3 in the current round.

each iteration, it performs similar operations including *Selection & Downloading*, *Random Reshuffling*, and *Re-encryption & Uploading* of index files. Finally, the desired data item and the selected dummy data items are downloaded, randomly reshuffled, re-encrypted and uploaded. Detailed explanations of the operations are presented next, with a simple example given in Fig. 3.

3.2.2 Selection of Dummy Data Items

When the user intents to retrieve a data item (denote its ID by $Q^0_{cur}.D_R$), it also requests the following dummy data items to conceal its intention:

- the first dummy (whose ID is denoted as $Q^0_{cur}.D_S$): the dummy that may swap its storage location with $Q^0_{cur}.D_R$ after access with a probability of 1/2;
- the second dummy (whose ID is denoted as $Q^0_{cur}.D_N$): the dummy that will not swap its storage location with others.

$Q^0_{cur}.D_S$ and $Q^0_{cur}.D_N$ are selected to make sure that the user's request at each round has the same format: *the user always requests three data locations, out of which two and only two of them are from the ones accessed in the previous round*. Note that requiring user's request at each round to have the same format is necessary to hide the true access pattern [10]. Specifically, it hides the information about whether user's requests at two rounds are intended for the same data item. Also note that the second dummy is needed in order to guarantee that each access can keep the

same format (please refer to [21] for detailed explanations). To maintain the same format in each access, the data structure Q^0_{pre} is downloaded from the server, which records the information about the data items (namely, the data IDs and their corresponding locations) accessed in the previous round. Then, the dummies for the current round are selected according to the following rules:

- For the first dummy (i.e., $Q^0_{cur}.D_S$): (i) If the intended data item is the same as the intended data item or the first dummy in the previous round, then the first dummy will be selected uniformly at random from the set of all data items excluding the intended data item of the current round. (ii) Otherwise, the first dummy will be randomly selected from the intended data item or the first dummy in the previous round with equal probability. (Refer to lines 3 to 7 in Step 1 as shown in Fig. 2.)
- For the second dummy (i.e., $Q^0_{cur}.D_N$), its selection depends on the selection results of the first dummy: (i) If both the intended data item and the first dummy have appeared in the previous round, the second dummy will be selected uniformly at random from the set of all data storage locations excluding the locations accessed in the previous round. (ii) Otherwise, the second dummy will be selected uniformly at random from the locations accessed in the previous round excluding locations of the already-selected files. (Refer to lines 12 to 20 in Step 2 as shown in Fig. 2 when $t = 0$.)

In the example given in Fig. 3, in the previous round, data #10 was intended by the user and data #1 was selected as the first dummy. Since data #3 is needed in the current round (i.e., case (ii) in the first dummy selection rules), the user randomly selects the first dummy, which is data #1 in this example, from data #10 and data #1 (as shown by step 3). As the selected data items did not both appear in the previous round (i.e., case (ii) in the second dummy selection rules), the second dummy's location, which is 7 (as shown by step 15), is selected from data #10 and data #9's locations (i.e., data locations #7 and #11).

3.2.3 Selection, Downloading, Processing and Uploading of Index Files

First, the single top-level index file is downloaded and decrypted, and its ID is recorded in $Q^T_{cur}.D_R$, $Q^T_{cur}.D_S$, and $Q^T_{cur}.D_N$, i.e., $Q^T_{cur}.D_R = Q^T_{cur}.D_S = Q^T_{cur}.D_N = 1$ (as shown by step 4 in the example of Fig. 3). Then, three index files for each level t, where $(T - 1) \geq t \geq 1$, are selected, downloaded, processed and uploaded, in an iterative and top-down manner. Without loss of generality, the following describes the operations for iteration t.

3.2.3.1 ***Selection & Downloading of Level-t Index Files***: The files that contain the level-t indices of the intended data item ($Q^0_{cur}.D_R$) and the first dummy ($Q^0_{cur}.D_S$) are first selected to access. The IDs of these files are denoted as $Q^t_{cur}.D_R$ and $Q^t_{cur}.D_S$ respectively. Note that, these file IDs can be found out by using the afore-defined $f(\cdot,\cdot)$ function, i.e., $Q^t_{cur}.D_R = f(Q^0_{cur}.D_R, t)$ and $Q^t_{cur}.D_S = f(Q^0_{cur}.D_S, t)$. Then, similar to the selection of dummy data items, additional dummy index files are selected to make sure that, in each round, three level-t index files are accessed and exactly two of them appeared in the previous round. The following rules are applied in the selection:

- For the first dummy index file (i.e., $Q^t_{cur}.D_S$): If the intended data item and the first dummy share the same level-t index file, the first dummy index file is re-selected uniformly at random from the index files whose storage locations are stored in files $Q^{t+1}_{cur}.D_R$or $Q^{t+1}_{cur}.D_S$, i.e., the level-$(t + 1)$ intended index file and the first dummy index file downloaded in the previous iteration of this algorithm. (Refer to lines 8 to 10 in Step 2 as shown in Fig. 2.)
- For the second dummy index file (i.e., $Q^t_{cur}.D_N$): (i) If the intended index file and the first dummy index file have both appeared in the previous round, the second dummy index file will be selected uniformly at random from all level-t index file locations excluding the locations that appeared in the previous round. (ii) Otherwise, the second dummy index file will be selected uniformly at random from the locations that appeared in the previous round excluding locations of the already-selected files. (Refer to lines 12 to 20 in Step 2 as shown in Fig. 2 when $t \neq 0$.)

After the level-t index files have been selected, the locations of files $Q^t_{cur}.D_R$ and $Q^t_{cur}.D_S$ can be found by searching their indices in the downloaded level-$(t + 1)$ index files, i.e., files $Q^{t+1}_{cur}.D_R$ and $Q^{t+1}_{cur}.D_S$. Then the locations of the three level-t index files are provided to the server and the files can be downloaded. Note that, the locations are presented to the server in an arbitrary order, so that the server cannot distinguish between desired index files and dummies. The downloaded files are then decrypted with the user's key.

In the example given in Fig. 3, since the intended data item and the first dummy share the same level-1 index file I^1_1, the user randomly selects a new first dummy index file, which is I^1_3 in this example, from level-1 index files $\{I^1_2, I^1_3, I^1_4\}$ (as shown by steps 6 and 7). Then the user looks up I^2_1 to find out the storage locations $Q^1_{cur}.L_R$ and $Q^1_{cur}.L_S$ (as shown by step 8). Since both I^1_1 and I^1_3 were accessed in the previous round, the user selects the second dummy index file with location #4 (as shown by step 9). Hence, the user retrieves the files from level-1 storage locations #2, #3 and #4.

3.2.3.2 ***Random Reshuffling of Selected Level-t Index Files***: The intended index file ($Q^t_{cur}.D_R$) and the first dummy index file ($Q^t_{cur}.D_S$) may swap their storage locations with a probability of 1/2. If the swap happens, the index information of these files should be updated in

their index files $Q_{cur}^{t+1}.D_R$ and $Q_{cur}^{t+1}.D_S$, respectively. In the example given in Fig. 3, since files $Q_{cur}^{1}.D_R$ and $Q_{cur}^{1}.D_S$ are swapped, the user updates I_1^2 accordingly (as shown by steps 11 and 12).

3.2.3.3 ***Re-encryption & Uploading of Index Files***: Now, we have completed the processing of level-(t + 1) index files $Q_{cur}^{t+1}.D_R$, $Q_{cur}^{t+1}.D_S$, and $Q_{cur}^{t+1}.D_N$. To hide content and/or location changes made to them, these files should be re-encrypted before being uploaded back to the server. In our scheme, re-encryption is performed by applying the Cipher Block Chaining (CBC) encryption techniques [20] on the file content, where the first block of the file is a non-reappearing nonce. The user's key is used inthe re-encryption. This way, the same secret key can be reused for encrypting all files, which simplifies the key management at the user. Such re-encryption process ensures that a computationally bounded adversary does not have non-negligible advantage at determining whether a pair of encrypted data items (before and after re-encryption, respectively) carry the same data content.

After re-encryption, files $Q_{cur}^{t+1}.D_R$, $Q_{cur}^{t+1}.D_S$, and $Q_{cur}^{t+1}.D_N$ are uploaded to their locations, respectively, but in an arbitrary order to make it difficult for the server to track these files. At the end of iteration t, data structure Q_{pre}^{t} should be replaced by Q_{cur}^{t}, then re-encrypted and uploaded to location $Hist[t]$. This way, next time when Q_{pre}^{t} is downloaded, it will reflect the mostly recent history.

In the example in Fig. 3, I_1^2 and Q_{cur}^{1} are re-encrypted and uploaded to the server at the storage locations #0 and $Hist[1]$, respectively (as shown by step 13).

3.2.4 Downloading, Processing and Uploading of Data Items

After the above steps, the level-1 index files have been downloaded and decrypted. Based on the index information in these files, the desired data item and two additional dummy data items can be downloaded from the server and decrypted with the user's key. Upon the user's access to the desired data item has been completed, the intended data item and the first dummy may swap their storage locations with a probability of 1/2, and if the swap happens, changes will be made to the level-1 index files $Q_{cur}^{1}.D_R$ and/or $Q_{cur}^{1}.D_S$, respectively. Finally, the three level-1 index files and the three data items are re-encrypted and uploaded to the server. Also, data structure Q_{pre}^{0} is updated to Q_{cur}^{0}, re-encrypted and uploaded to the server. The re-encryption and uploading operations are performed in the similar manner as described above.

In the example given in Fig. 3, the user looks up I_1^1 to find the storage locations $Q_{cur}^{0}.L_R = 5$ and $Q_{cur}^{0}.L_S = 4$. As afore-explained, the user selects the second dummy's storage location $Q_{cur}^{0}.L_R = 7$ (as shown by steps 14 and 15). Since data items $Q_{cur}^{0}.D_R$ and $Q_{cur}^{0}.D_S$ are swapped, the content of I_1^1 is updated (as shown by steps 17 and 18). Finally, the re-encrypted level-1 index files, Q_{cur}^{0} and data items are uploaded to the server respectively.

Remark: Note that there is not need to include the second dummy data items/index files in the reshuffling process. This is because the server does not know which two data items/index files are involved in the reshuffling, because all three files are re-encrypted before uploading.

4. Security Analysis

In this section, we show that the proposed scheme can preserve the privacy of user's access pattern in the long run. That is, after a sufficiently large number of accesses, the frequency with which each data item has been accessed cannot be figured out by the server. Then we discuss the practical implications of this security property through analyzing how our scheme can deal with some typical attacks that are based on the knowledge of access pattern.

4.1 Access Pattern of Index Files

We first show that the access pattern of index file locations, which can be observed by the server, does not reveal extra information about the data access pattern. In the proposed scheme, index files are used to facilitate user query and data access. The content of an index file is protected by being re-encrypted after each access, based on the user's secret key and a random non-repeating nonce. Hence, it is impossible for the server to gain information about the data access pattern from the content of index files. The following theorem states that observing the access pattern of index file storage locations does not reveal more information about data access pattern than observing only the access pattern of data storage locations.

Theorem 4.1: The storage server cannot gain any advantage in inferring user's data access pattern through observing the access pattern of index file storage locations.

Proof: Due to space limitation, the proof of Theorem 4.1 is omitted in this paper. Please refer to our full technical report [21] for details. □

4.2 Access Pattern of Data Items

As the observed access pattern of index file locations does not help in inferring data access pattern, we next study what can be inferred from observing only the access pattern of data storage locations. The following theorem formally states the property that, if the server can only observe the access pattern of data storage locations, the data access pattern, namely, the data item requested by the user and the frequency with which each data item has been accessed by the user, can be preserved in the long run.

Theorem 4.2: If a user has accessed the data items, despite the user access sequence, for a sufficiently large number of times, each storage location at the server is accessed uniformly at random.

Proof: The proof of Theorem 4.2 has been reported in

the preliminary version of this paper [34], thus is omitted in this paper. Please refer to [34] for details. □

4.3 Discussion

To further understand the practical implications of the above security properties, In the following, we discuss a few typical attacks that are based on the knowledge of data access pattern, and analyze how our scheme can deal with the attacks.

4.3.1 Security against Tracking Data Items

Suppose the server has identified a particular user data item via other means, e.g., physical spying. It may want to keep track of this data item thereafter. Using our proposed scheme, due to the property described in Theorem 4.2, after a sufficiently large number of accesses, the server does not have non-negligible advantage at determining which location the target data item is at. For example, after the first round that the target item has been accessed, from the server's perspective, the target item may be stored at any of the three accessed locations with an equal probability of 1/3. Then if any of these three locations is accessed in the next round, the probability will be divided further among the newly accessed locations. Therefore, by solely observing the storage locations accessed by the user, the server could lose track of the target data item quickly.

4.3.2 Security against Focused Attacks on Selected Data Items

Some of the user's data items may be requested with very high frequency. These files are often important to the user. If a malicious server knows which data items are frequently accessed, it may launch intensive attacks on the data, attempting to find out the content or contextual information of the data. Note that, such attacks are sometimes feasible in practice, for example, when the adopted data encryption algorithm or the key chosen by the user is not sophisticated enough, or some side information about the data can be obtained in other means. Using our proposed scheme, due to the property described in Theorem 4.2, all data storage locations will be equally accessed in the long run. Hence, the server cannot identify which data items are frequently requested by the user. Similarly, some of the user's data items may be requested with very low frequency, e.g., backup data. A malicious server may want to stealthily delete these rarely-accessed user data items to save storage and maintenance cost for itself without being noticed by the user. Such attack can also be stopped as our proposed scheme prevents the server from identifying rarely requested data items.

5. Analysis of Convergence Rate

As shown in the previous section, using the proposed scheme, every data item is uniformly randomly distributed to all storage locations after a sufficiently large number of accesses. In this section, we analyze the convergence rate of the proposed scheme. Specifically, we are interested in finding how many accesses are needed before every data item become uniformly randomly distributed to all storage locations. Apparently, different true access patterns may result in different convergence rates. In this paper, we will present the analysis of the convergence rate under one particular access pattern, in which *the user always requests the same data item as its true target*. The convergence rates for more complicate access patterns are much more difficult to analyze and we will investigate them in our future work. Nevertheless, the analysis of convergence rate we present in this paper will provide some insights about how fast the proposed scheme converges.

In the following analysis, we model the user's access process as a Markovian process, denoted as MC-1. Specifically, each state in MC-1 is a permutation of (d_1, ⋯ , d_n), which stands for one distribution of the n data items to n storage locations. For simplicity, we assume that data item d_1 is the target data item in each access, i.e., $D_R = d_1$. Note that because the location of D_N does not change in each access, the selection of D_N does not affect the distribution of data items' storage locations. Hence, we only consider the behavior of D_R and D_S in the following analysis. Given the assumption that $D_R = d_1$ in each access, then D_S*is selected uniformly at random from all the data items, excluding* d_1 (see Section 3.2). In other words, D_S may be any data item from $\{d_2, \cdots, d_n\}$ with equal probability, which is $1/(n-1)$.

5.1 Preliminaries

In literature, many methods have been proposed to study the convergence rate of a Markovian process [31-33]. Our analysis of the convergence rate is based on the relation between the convergence rate and the second largest eigenvalue of the transition matrix. Specifically, our analysis of MC-1's convergence rate is based on the following fact.

Theorem 5.1: Given a Markovian process with initial state vector $\alpha_0 = \{\alpha_{0,1}, \alpha_{0,2}, \cdots, \alpha_{0,n}\}$, *the state vector after* m *steps* $\alpha_m = \{\alpha_{m,1}, \alpha_{m,2}, \cdots, \alpha_{m,n}\}$ *and the steady state vector* $\pi = \{\pi_1, \cdots, \pi_n\}$, *it has*

$$sup_{j\in E}\left|\alpha_{m,j} - \pi_j\right| = O(\left|\lambda_2\right|^m), \tag{1}$$

where $E = \{1, \cdots, n\}$, *and* $\lambda_2 < 1$ *is the second largest eigenvalue of the transition matrix.*

Eq. (1) implies that, for a Markovian process, the convergence rate from an arbitrary initial state to the steady state is upper-bounded by the rate that $|\lambda_2|^m$ approaches zero, where m is the number of steps. For example, suppose the system requires that $sup_{j\in E}|\alpha_{m,j} - \pi_j| \le \varepsilon$, where $\varepsilon > 0$. If we let

$$c\left|\lambda_2\right|^m \le \varepsilon, \tag{2}$$

which is equivalent to

$$m \ge \log_{\lambda_2}\varepsilon - \log_{\lambda_2} c = O(\log_{\lambda_2}\varepsilon), \tag{3}$$

then we will get $sup_{j\in E}|\alpha_{m,j} - \pi_j| \le \varepsilon$ after $O(\log_{\lambda 2}\varepsilon)$ steps. In our following analysis, we will firstly find λ_2 for the transition matrix of MC-1. Then we find an upper-bound for the convergence rate of MC-1 based on the value of λ_2.

5.2 The Transition Matrix of MC-1

Denote each state of MC-1 as σ_i (i = 1, ⋯ , n!), which stands for one distribution of the n data items to n storage locations. Then according to the scheme, the transition function $g(\sigma_i, \sigma_j)$ from a state σ_i to another state σ_j is defined as following:

- $g(\sigma_i, \sigma_j)$ = 1/2, if σ_i= σ_j, which means that each data item's storage location does not change after one access. This situation is resulted from D_R and $D_S$not swapping their locations at the end of the access. Note that D_R= d_1 and there are n − 1 possible choices of D_S and the probability of swapping is 1/2, thus $g(\sigma_i, \sigma_j)$ = $\frac{1}{2}\cdot\frac{1}{n-1}\cdot(n-1) = 1/2$.
- $g(\sigma_i, \sigma_j) = \frac{1}{2(n-1)}$, if σ_i and σ_j differs only in d_1's location and one of the rest of data items location. Inthis case, it means that D_R and D_S swap their locations at the end of the access. For example, suppose σ_i= (d_1, d_2, d_3, ⋯ , d_n) and σ_j= (d_2, d_1, d_3, ⋯ , d_n), then σ_i may transit to σ_j if d_2 is selected as the first dummy data item and d_1 and d_2 swap their locations at the end of the access.
- Otherwise, $g(\sigma_i, \sigma_j)$ = 0. For example, suppose σ_i = (d_2, d_3, d_1, d_4, ⋯ , d_n) and σ_j = (d_3, d_2, d_1, d_4, ⋯ , d_n), then $f(\sigma_i, \sigma_j)$ = 0, because there is no way that d_3 and d_2 swap their locations in one access while d_1 is the true target.

Note that in the proposed scheme, transitions between two states are symmetric, i.e., $g(\sigma_i, \sigma_j) = g(\sigma_j, \sigma_i)$.

As an example, let n = 3, thus there are in total 3! = 6 states in MC-1. Note that in practice, if n = 3, our scheme will always retrieve the whole database. However, without loss of generality, we use n = 3 to simplify the presentation. Fig. 4 shows the transition matrix for this process, from which we can get the transition probability between different states. For example, Fig. 4 reads that the transition probability between states (1 2 3) and (2 1 3) is 1/4, which is $\frac{1}{2}\cdot\frac{1}{3-1}$.

It is easy to calculate that, when n = 3, the transition matrix has the second largest eigenvalue λ_2 = 3/4. Therefore, according to Eq. (1), the rate at which an arbitrary state converges to the steady state is upper-bounded by the rate at which $\left(\frac{3}{4}\right)^m$ approaches zero as m (the number of accesses) increases. As afore-explained in Section 5.1, we will get $sup_{j\in E}|\alpha_{m,j} - \pi_j| \le \varepsilon$, where $\varepsilon > 0$, after $O\left(\log_{\frac{3}{4}}\varepsilon\right)$ steps (i.e., data accesses). For example, if we let $\varepsilon = 1/n^c$, the process will converge in $O\left(\log_{\frac{3}{4}}\frac{1}{n^c}\right) = O\left(c\log_{\frac{3}{4}} n\right) = O(\log n)$ steps.

State	(1 2 3)	(2 1 3)	(1 3 2)	(2 3 1)	(3 1 2)	(3 2 1)
(1 2 3)	1/2	1/4	0	0	0	1/4
(2 1 3)	1/4	1/2	0	1/4	0	0
(1 3 2)	0	0	1/2	1/4	1/4	0
(2 3 1)	0	1/4	1/4	1/2	0	0
(3 1 2)	0	0	1/4	0	1/2	1/4
(3 2 1)	1/4	0	0	0	1/4	1/2

Fig. 4. Example transition probabilities between states when n = 3. Each parenthesized sequence of numbers represents a distribution of data items to different locations. For example, (1 2 3) denotes that data item d_1, d_2, and d_3 are stored in locations 1, 2, and 3 respectively.

5.3 The 2nd Largest Eigenvalue of MC-1

Before we present the analysis of the second largest eigenvalue of the transition matrix, we describe how we organize the transition matrix to simplify the proof. Apparently, the order in which different states appear in the rows and columns of the transition matrix affects the representation of the transition matrix. In the following, we describe our way of organizing the transition matrix, by explaining how to get the transition matrix for a system of n + 1 data items (denoted as Π_{n+1}) based on the transition matrix of n data items (denoted as Π_n). Specifically, Π_{n+1} can be considered as the addition of two matrices, namely, matrix $\Pi_{n+1,A}$ which is derived from Π_n, and matrix $\Pi_{n+1,B}$. Fig. 5 shows a high-level illustration of these two matrices. Specifically:

- In $\Pi_{n+1,A}$(as shown in Fig. 5(a)), the sub-matrices corresponding to states from the same groups are the same as Π_n multiplying $\frac{n-1}{n}$. The rest of the entries in $\Pi_{n+1,A}$ are all zero.
- In $\Pi_{n+1,B}$(as shown in Fig. 5(b)), the diagonal entries are all $\frac{1}{2n}$. In addition, for each row/column, there is only one non-zero entry besides the diagonal entry, which is also $\frac{1}{2n}$.

We claim that the second largest eigenvalue, λ_2, for the transition matrix that we described previously is $1 - \frac{1}{2(n-1)}$, where n is the number of data items. The rest of this section presents the proof of this claim. We firstly prove in Section 5.3.1 that $1 - \frac{1}{2(n-1)}$ is one of the

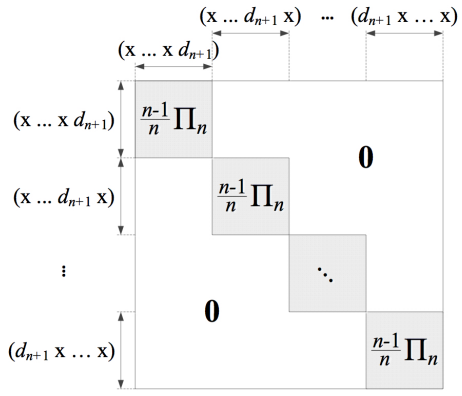


Fig. 5. Illustration of $\Pi_{n+1,A}$ and $\Pi_{n+1,B}$.

eigenvalues for the transition matrix with n data items. We then show in Section 5.3.2 that $1-\frac{1}{2(n-1)}$ is indeed the second largest eigenvalue of the transition matrix.

5.3.1 Eigenvalues of the Transition Matrix

Theorem 5.2: For a system with n data items, the afore-described transition matrix of MC-1 has one of its eigenvalues λ_n *as* $1-\frac{1}{2(n-1)}$, *with one of its corresponding eigenvector* $\boldsymbol{V_n}$ *as the following:*

$$\boldsymbol{V_n}=\begin{cases}\{0,1,0,1,-1,-1\}, & \text{if } n=3\\ \{\{\boldsymbol{V_{n-1}}\}_{n-1},\boldsymbol{0_{n-1}}\}, & \text{if } n>3\end{cases} \tag{4}$$

where $\{\boldsymbol{V_{n-1}}\}_{n-1}$ *is the concatenation of* $n-1$ *copies of* $\boldsymbol{V_{n-1}}$, *and* $\boldsymbol{0_{n-1}}$ *is an* $((n-1)!)$*-entry zero vector.*

Proof: We prove the theorem by induction. Let $\Pi_n$denote the transition matrix of interest. Generally, for an eigenvalue λ_n with its eigenvector $\boldsymbol{V_n}$, we have

$$(\Pi_n-\lambda_n I_n)\boldsymbol{V_n}=\boldsymbol{0_n}, \tag{5}$$

where I_n stands for the $(n!)\times(n!)$ identity matrix and $\boldsymbol{0_n}$ is an $(n!)$-entry zero vector.

It is easy to verify that when $n=3$, $\{0, 1, 0, 1, -1, -1\}$ is an eigenvector with eigenvalue 3/4 for the matrix shown in Fig. 4. In the following, we will show that $\boldsymbol{V_n}=\{\{\boldsymbol{V_{n-1}}\}_{n-1}, \boldsymbol{0_{n-1}}\}$ when $n>3$.

Suppose $\boldsymbol{V_n}$ is an eigenvector in the format of Eq. (4), with eigenvalue $\lambda_n=1-\frac{1}{2(n-1)}$. As afore-explained,

$$\Pi_{n+1}=\Pi_{n+1,A}+\Pi_{n+1,B}. \tag{6}$$

Given that $\lambda_n=1-\frac{1}{2(n-1)}$ and $\lambda_{n+1}=1-\frac{1}{2n}$, therefore,

$$\begin{aligned}&(\Pi_{n+1}-\lambda_{n+1}I_{n+1})\boldsymbol{V_{n+1}}\\ &=\left(\left(\Pi_{n+1,A}-\frac{n-1}{n}\lambda_n I_{n+1}\right)+\left(\Pi_{n+1,B}-\frac{1}{n}I_{n+1}\right)\right)\boldsymbol{V_{n+1}}\\ &=\left(\Pi_{n+1,A}-\frac{n-1}{n}\lambda_n I_{n+1}\right)\boldsymbol{V_{n+1}}+\left(\Pi_{n+1,B}-\frac{1}{n}I_{n+1}\right)\boldsymbol{V_{n+1}}.\end{aligned} \tag{7}$$

Based on the relation between $\Pi_{n+1,A}$ and Π_n (as shown in Fig. 5(a)), and given that Eq. (5) holds and $\boldsymbol{V_{n+1}}=\{\{\boldsymbol{V_n}\}_n, \boldsymbol{0_n}\}$, it is easy to see that the first item of Eq. (7) is equal to $\boldsymbol{0_{n+1}}$, i.e.,

$$(\Pi_{n+1,A}-\frac{n-1}{n}\lambda_n I_{n+1})\boldsymbol{V_{n+1}}=\boldsymbol{0_{n+1}}. \tag{8}$$

Now we show that the second item of Eq. (7), i.e., $(\Pi_{n+1,B}-\frac{1}{n}I_{n+1})\ \boldsymbol{V_{n+1}}$, is also equal to $\boldsymbol{0_{n+1}}$. Let $\Pi'_{n+1,B}=(\Pi_{n+1,B}-\frac{1}{n}I_{n+1})$. From Fig. 5(b), we can see that $\Pi'_{n+1,B}$ differs from $\Pi_{n+1,B}$ only in the diagonal entries: diagonal entries in $\Pi'_{n+1,B}$ are $-\frac{1}{2n}$ whereas diagonal entries in $\Pi_{n+1,B}$ are $\frac{1}{2n}$. Because each row of $\Pi'_{n+1,B}$ only has two non-zero entries, we observe the following facts when scalar-multiplying each row of $\Pi'_{n+1,B}$ with $\boldsymbol{V_{n+1}}$: the diagonal entry and the other non-zero entry are multiplying the same value, either 0, 1 or -1. Specifically,

- If the diagonal entry is multiplying 0, the other non-zero entry of this row is also multiplying 0.
- If the diagonal entry is multiplying 1, the other non-zero entry of this row is also multiplying 1.
- If the diagonal entry is multiplying -1, the other non-zero entry of this row is also multiplying -1.

As a result, the scalar multiplication of each row of $\Pi'_{n+1,B}$ and $\boldsymbol{V_{n+1}}$ is 0, which means that

$$(\Pi_{n+1,B}-\frac{1}{n}I_{n+1})\boldsymbol{V_{n+1}}=\boldsymbol{0_{n+1}}. \tag{9}$$

Therefore, Eq. (7) is equal to $\mathbf{0}_{n+1}$. As a result, $\lambda_{n+1} = 1 - \frac{1}{2n}$ is an eigenvalue for Π_{n+1} and its eigenvector $\boldsymbol{V}_{n+1} = \{\{\boldsymbol{V}_n\}_n, \mathbf{0}_n\}$. □

5.3.2 The 2nd Largest Eigenvalue

Theorem 5.3: If $\lambda' > 1 - \frac{1}{2(n-1)}$ *is an eigenvalue for the afore-described transition matrix of MC-1 with n data items, then* $\lambda' = 1$.

Proof Sketch: We will also use induction to prove the theorem. Due to space limitation, we only present a sketch of the proof. Please refer to [21] for details.

Firstly, as shown in Section 5.2, when $n = 3$, the only eigenvalue that is greater than $1 - \frac{1}{2(n-1)} = \frac{3}{4}$ is 1.

Now assume $\lambda_n = 1 - \frac{1}{2(n-1)}$ is the second largest eigenvalue for Π_n, while there exists $\varepsilon > 0$ such that $\lambda'_{n+1} = 1 - \frac{1}{2n} + \varepsilon$ is the second largest eigenvalue (i.e., $\lambda'_{n+1} < 1$) for Π_{n+1}. Denote one of its corresponding eigenvectors as

$$\boldsymbol{V}'_{n+1} = \{v_1, v_2, \cdots, v_m\}, \tag{10}$$

where $m = (n+1)!$. Therefore, we have

$$(\Pi_{n+1} - \lambda'_{n+1} I_{n+1}) \boldsymbol{V}'_{n+1} = \mathbf{0}_{n+1}, \tag{11}$$

which is equivalent to

$$\begin{aligned}&(\Pi_{n+1} - \lambda'_{n+1} I_{n+1}) \boldsymbol{V}'_{n+1} \\ &= (\Pi_{n+1,A} - \frac{n-1}{n}(\lambda_n + \varepsilon') I_{n+1}) \boldsymbol{V}'_{n+1} + (\Pi_{n+1,B} - \frac{1}{n} I_{n+1}) \boldsymbol{V}'_{n+1} \\ &= \mathbf{0}_{n+1}.\end{aligned} \tag{12}$$

We have shown in [21] that in the second item of the above equation, i.e., $(\Pi_{n+1,B} - \frac{1}{n} I_{n+1})\ \boldsymbol{V}'_{n+1}$, there is one group whose summation is $\mathbf{0}_n$. Given that the overall summation equals $\mathbf{0}_{n+1}$, there must be one group in $(\Pi_{n+1,A} - \frac{n-1}{n}(\lambda_n + \varepsilon') I_{n+1}) \boldsymbol{V}'_{n+1}$, whose result is also $\mathbf{0}_n$. This mean that there exists an eigenvalue $\lambda_n = 1 - \frac{1}{2(n-1)} + \varepsilon'$ for Π_n, which contradicts to our assumption. □

5.4 Analysis of the Convergence Rate

Now given the second largest eigenvalue of the transition matrix, we now can analyze the convergence rate of MC-1 based on Theorem 5.1. Specifically, suppose we define convergence as

$$sup_{j \in E} \left| \alpha_{m,j} - \pi_j \right| = O(|\lambda_2|^m) = c|\lambda_2|^m < n^{-\varepsilon}, \tag{13}$$

where c is a constant, $\varepsilon > 0$ is a benchmark for convergence, and n is the number of data items. On the other hand,

$$\left(1 - \frac{1}{2(n-1)}\right)^m = \left(\left(1 - \frac{1}{2(n-1)}\right)^{2(n-1)}\right)^{\frac{m}{2(n-1)}} \le \left(\frac{1}{e}\right)^{\frac{m}{2(n-1)}}. \tag{14}$$

Let $m = t \times 2(n-1)$, then

$$\left(1 - \frac{1}{2(n-1)}\right)^m \le \left(\frac{1}{e}\right)^t. \tag{15}$$

As a result, MC-1 converges if we have

$$c\left(\frac{1}{e}\right)^t < n^{-\varepsilon}, \tag{16}$$

which is equivalent to

$$t > \varepsilon \ln n + \ln c. \tag{17}$$

Therefore, to converge, m need to satisfy the following

$$m = t \times 2(n-1) > (\varepsilon \ln n + \ln c) \times 2(n-1) = O(\varepsilon n \ln n). \tag{18}$$

Thus, it takes $O(\varepsilon n \ln n)$ steps for MC-1 to converge. In other words, the user need to make $O(\varepsilon n \ln n)$ accesses before the access appears to be uniformly at random.

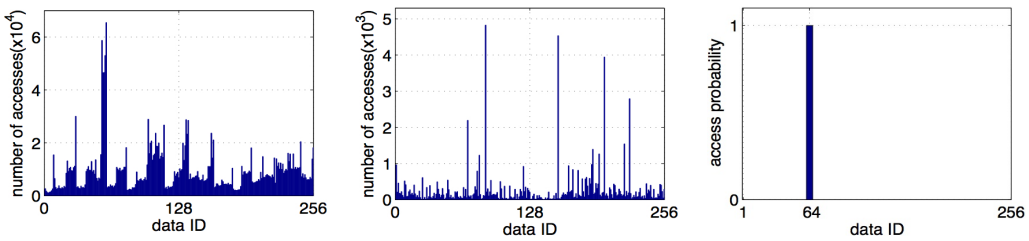
6. Performance Evaluation

6.1 Evaluation Setup

To evaluate the performance of the proposed scheme, we have collected two user access traces from two popular cloud service providers: YouTube [22] and Baidu [23]. As shown in Figs. 6(a) and (b), both the YouTube user and the Baidu user have 256 files stored at the server. Different files have been accessed with different frequencies over time. Moreover, we have created an additional user who always requests the same file from the server, called the SFA (Single File Access) user, as shown in Fig. 6(c). We use the SFA user to emulate an extreme access pattern. The total number of files for the SFA user is also 256.

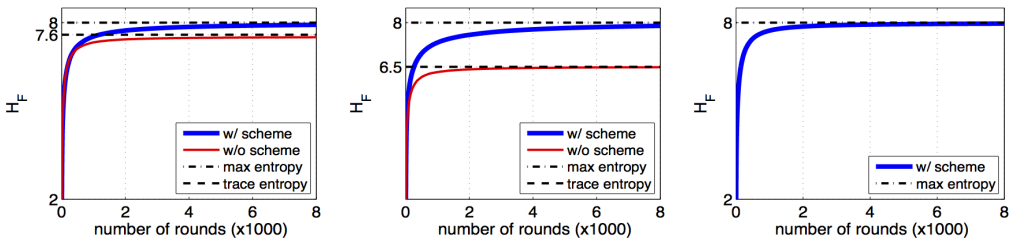
6.2 Access Frequency Privacy

To study how well our proposed scheme preserves a cloud user's access frequency privacy, we propose to use *entropy* to measure the distribution of the user's access frequencies to different files. Specifically, let C_i denote the number of accesses to the file stored at storage location i.



(a) YouTube (b) Baidu (c) SFA

Fig. 6. Data access traces and distribution used in the performance evaluation.



(a) YouTube (b) Baidu (c) SFA

Fig. 7. The entropy of access frequency vs. the number of access rounds for a particular simulation run under different access scenarios. In (c), because the SFA user always requests the same data item at each round, the entropy of access frequency without using our proposed scheme is always zero, which is not shown in the figure.

Then, the access frequency to location i is $F_i = \frac{c_i}{\sum_j c_j}$, and the entropy of access frequency is $H_F = -\sum_i F_i \log(F_i)$. For example, H_F of the YouTube and Baidu traces is around 7.6 and 6.5, respectively, which can be calculated by counting the number of accesses to each file in Figs. 6(a) and (b). Clearly, for a given set of files stored at the server, the maximum entropy is achieved when all file locations have been accessed with an equal probability. This means that, the maximum entropy for accessing 256 files is $H_F^{\max}(256) = -256 \times \frac{1}{256}\log\left(\frac{1}{256}\right) = 8$.

We evaluate how the entropy of access frequency changes as the number of access rounds increases. Fig. 7 plots the results (averaged over 100 simulation runs) for different access scenarios. It can be seen clearly from the figures that, with our scheme, the entropy of access frequency improves over the original trace, and converges gradually to the maximum entropy in all simulated scenarios. This confirms our analytical study in Section 4 and Theorem 4.2 that the access frequency distribution converges towards the uniform distribution in the long run.

6.3 Data Item's Location Privacy

As discussed in Section 4, when the user employs our proposed scheme, the cloud server loses track of a certain data item gradually over time. In other words, from the server's perspective, the uncertainty of a data item's storage location increases gradually over time. Similar to the evaluation of access frequency privacy, we also use *entropy* to measure the uncertainty of a particular data item's location from the server's perspective. It is defined as $H_L = -\sum_i p_i \log(p_i)$, where $p_i$is the probability that the data item is at location i from the server's perspective. We evaluate how the entropy of the data item's location distribution grows as the number of access rounds increases. For each access scenario, we collect the statistics of the most accessed data item and the least accessed data item, and results (averaged over 100 simulation runs) are plotted in Figs. 8(a) and (b), respectively. From the figures, we can see that a data item's location distribution entropy reaches the maximum regardless of their real access frequency. Note that, without our proposed scheme, a data item's location distribution entropy is zero because its location is fixed and known to the server. Also note that based on the analysis in Section 5, the convergence rate for SFA is $O(\varepsilon n \ln n) = O(\varepsilon \cdot 256 \ln 256) \approx O(1400\varepsilon)$. From Fig. 8(a), we can see the curve for SFA converges to max entropy after around 1500 steps, which confirms our analysis of the convergence rate in Section 5.

7. Overhead Analysis

7.1 Communication and Computational Overhead

With our proposed scheme, to access a single data item, the cloud user needs to obtain the following information from the cloud server:

- Three index files at each level of the storage hierarchy; each index file records the storage locations of m index files at its next lower level and it takes log

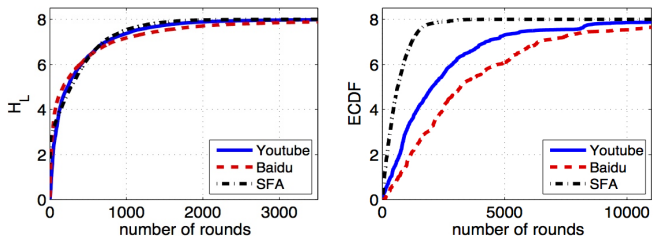


Fig. 8. (a) Average entropy of location distribution vs. the number of access rounds for the most frequently requested data item, (b) Average entropy of location distribution vs. the number of access rounds for the least frequently requested data item.

n bits to represent a storage location.

- Three index files at each level of the storage hierarchy; each index file records the storage locations of m index files at its next lower level and it takes log n bits to represent a storage location.
- One access history file at each level of the storage hierarchy; each access history file records the IDs and storage locations of three index files (at this level) that were accessed in the previous round; hence, it contains six fields and each field is log n-bit long.
- The desired data item and two dummy data items; let τ denote the size of each data item in bits.

Recall that there is a total of $\log_m n$ levels in our proposed hierarchical storage structure. Therefore, the overall communication and computational overhead for accessing a single data item can be calculated as:

$$\mathrm{OH}_{c\&c} = m\log n \cdot 3\log_m n + 6\log n \cdot \log_m n + 3\tau. \quad (19)$$

It is easy to verify that:

$$\begin{cases} \min \mathrm{OH}_{c\&c} = \mathrm{OH}_{c\&c}\big|_{m=4} = 9(\log n)^2 + 3\tau; \\ \max \mathrm{OH}_{c\&c} = \mathrm{OH}_{c\&c}\big|_{m=n} = (3n+6)\log n + 3\tau. \end{cases} \quad (20)$$

7.2 Storage Overhead

As explained in Section 3.1, the total number of index files in our proposed scheme is $\frac{n-1}{m-1}$. Each index file recordsthe storage locations of m index files at its next lower level and it takes log n bits to represent a storage location. Therefore, the overall storage overhead at the cloud server can be calculated as:

$$\mathrm{OH}_{\text{s\_server}} = m\log n \cdot \frac{n-1}{m-1} + n\tau. \quad (21)$$

It is easy to verify that:

$$\begin{cases} \min \mathrm{OH}_{\text{s\_server}} = \mathrm{OH}_{\text{s\_server}}\big|_{m=n} = n\log n + n\tau; \\ \max \mathrm{OH}_{\text{s\_server}} = \mathrm{OH}_{\text{s\_server}}\big|_{m=2} = 2(n-1)\log n + n\tau. \end{cases} \quad (22)$$

At the user side, to operate our proposed scheme, the cloud user needs to store one access history file, three index files, and three more index files or data items at any given time. Thus the required storage at the user side is:

$$\mathrm{OH}_{\text{s\_user}} = 6\log n + 3m\log n + \max\{3m\log n,\ 3\tau\}. \quad (23)$$

7.3 Overhead Comparison

Based on the above overhead analysis, we set $m = 4$ in our scheme. In Table 2, we compare our scheme with one of the state-of-the-art access pattern preservation schemes for single-cloud-server systems [16].

Table 2. Overhead Comparison

	Our Scheme (m=4)	Scheme in [16]
Comm./Comp	$O((\log n)^2 + \tau)$	$O(\log n \cdot \tau)$
Storage (server)	$O(n \max\{\log n, \tau\})$	$O(n \cdot \tau)$
Storage (user)	$O(\max\{\log n, \tau\})$	$O(\sqrt{n} \cdot \tau)$

In practical cloud storage applications, the size of a data item (τ, in bits) is usually larger than log n bits, where n is the total number of data items. E.g., [16] proposes the data item size to be 64 KB, i.e., 2^{16} bits. It is generally impractical for a user to have more than $2^{2^{16}}$ data items. Under this assumption, it is interesting to see that our scheme is more efficient. Specifically, our scheme (i) consumes similar storage space at the cloud server; (ii) usually incurs significantly less communication and computational overhead; and (iii) requires significantly less storage space at the cloud user, which facilitates the employment of our proposed scheme on thin user devices such as mobile phones. Note that the better efficiency performance of our scheme is achieved under a less stringent privacy requirement than [16]; instead of requiring strict privacy protection to the data access pattern, our scheme aims to protect the privacy of the data access pattern in the long run.

8. Related Work

Although many schemes [4, 5, 24] have been proposed

to protect data confidentiality and data integrity for the cloud computing paradigm, little effort has been made to protect users' access pattern privacy. Private Information Retrieval (PIR) [9, 25, 26], Oblivious RAM [10-19] and Steganographic File Systems (SFS) [27-29] are the works most related to our solution.

Private Information Retrieval: PIR schemes aim to allow clients to retrieve information from a database while maintaining the privacy of the queries to the database. Fully implementing the PIR notion is, however, expensive. As shown by Sion *et al*. [9], deployment of any single-server PIR protocol is not necessarily more efficient than a simple transfer of the entire database due to computational costs. On the other hand, PIR schemes typically do not address data confidentiality, which makes PIR schemes unsuitable to be applied in the un-trusted cloud environments.

Oblivious RAM: In order to prevent the users' access pattern from being revealed, Oblivious RAM (ORAM) [10-19] has been proposed. In a latest version of ORAM, Stefanov *et al.* [16] proposed a framework for practical ORAM that significantly reduces the communication overhead in practical scenarios (e.g., $2^{20}<n<2^{35}$, where n is the number of data items). In Section 7, we have shown that our scheme is much more efficient in terms communication, computational and storage overheads in practical cloud storage applications under a less stringent privacy requirement.

Steganographic File Systems: Research efforts on steganographic file systems [27-29] are also related to our proposed design. The major differences lie in that, the research on SFS targets at protecting the information about existence and/or locations of sensitive files through hiding both short-term and long-term access patterns, while our proposal mainly targets at protecting long-term access pattern at low cost.

There is a concurrent effort [30] that addresses a similar problem as the one in our work. Their solution and ours share similar high-level ideas such as usage of dummies, hierarchical storage structure and file reshuffling. However, there are several key differences between the two solutions. E.g., our solution yields provable security and overhead performances and does not require user-side LRU cache or an empirical statistical access model.

9. Conclusions and Future Work

In this paper, we present a lightweight solution to the preservation data access pattern privacy in un-trusted storage. Rigorous proofs have been provided to show that the proposed scheme can provide full protection to data access pattern privacy in the long run. Detailed analysis and extensive evaluations have also been conducted to show that the scheme can protect the data access pattern privacy effectively after a reasonable number of accesses have been made. In the future work, we plan to enhance the scheme such that it can support private and efficient data updates, including data changes, data insertions and data deletions.

Acknowledgement

The research reported in this paper was supported in part by the Information Infrastructure Institute (iCube) of Iowa State University, the Security and Software Engineering Research Center (S2ERC), the National Science Foundation under Grants CNS 0831874 and CNS 0716744, and the office of Naval Research under Grant N000140910748.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing" Tech. Rep. UCB-EECS, 2009. Article (CrossRef Link)

[2] P. Mell and T. Grance, "Draft: Nist working definition of cloud computing" 2010. Article (CrossRef Link)

[3] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing" in Proc. SOSP, 2003. Article (CrossRef Link)

[4] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing" in Proc. IWQoS, 2009. Article (CrossRef Link)

[5] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained access control in cloud computing" in Proc. INFOCOM, 2010. Article (CrossRef Link)

[6] B. Chor and N. Gilboa, "Computationally private information retrieval" in Proc. STOC, 1997. Article (CrossRef Link)

[7] R. Ostrovsky and V. Shoup, "Private information storage" in Proc. STOC, 1997. Article (CrossRef Link)

[8] G. Itkis, "Personal communication, via oded goldreich" 1996.

[9] R. Sion and B. Carbunar, "On the computational practicality of private information retrieval" in Proc. NDSS, 2007. Article (CrossRef Link)

[10] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAM" in JACM, 1996. Article (CrossRef Link)

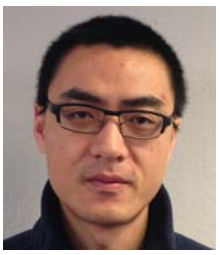
[11] P. Williams, R. Sion, and B. Carbunar, "Building castles out of mud: practical access pattern privacy and correctness on untrusted storage" in Proc. CCS, 2008. Article (CrossRef Link)

[12] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li, "Oblivious RAM with O((log n)^3) worst-case cost," in Proc. ASIACRYPT, 2011. Article (CrossRef Link)

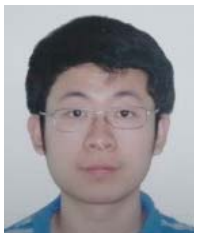
[13] M. T. Goodrich and M. Mitzenmacher, "Privacy-preserving access of outsourced data via oblivious RAM simulation" in Proc. ICALP, 2011. Article (CrossRef Link)

[14] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, "Oblivious RAM simulation with efficient worst-case access overhead," in Proc. CCSW, 2011. Article (CrossRef Link)

[15] E. Kushilevitz, S. Lu, and R. Ostrovsky, "On the (in)security of hash-based oblivious RAM and a new balancing scheme" in Proc. SODA, 2012. Article (CrossRef Link)
[16] E. Stefanov, E. Shi, and D. Song, "Towards practical oblivious RAM" in Proc. NDSS, 2012. Article (CrossRef Link)
[17] P. Williams and R. Sion, "Privatefs: A parallel oblivious file system" in Proc. CCS, 2012. Article (CrossRef Link)
[18] P. Williams and R. Sion, "Single round access privacy on outsourced storage" in Proc. CCS, 2012. Article (CrossRef Link)
[19] E. Stefanov and E. Shi, "Oblivistore: High performance oblivious cloud storage" in Proc. S&P, 2013. Article (CrossRef Link)
[20] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography. CRC Press, 1996. Article (CrossRef Link)
[21] K. Yang, J. Zhang, W. Zhang, and D. Qiao, "A lightweight solution to preservation of access pattern privacy in un-trusted clouds" Technical Report, 2012. [Online]. Available: Article (CrossRef Link)
[22] Youtube.[Online].Available: Article (CrossRef Link)
[23] Baidu. [Online]. Available: Article (CrossRef Link)
[24] C. Wang, Q. Wang, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data" in Proc. ICDCS, 2010. Article (CrossRef Link)
[25] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval" in Proc. FOCS, 1998. Article (CrossRef Link)
[26] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: Single database, computationally-private information retrieval" in Proc. IEEE Symposium on Foundations of Computer Science, 1997. Article (CrossRef Link)
[27] X. Zhou, H. Pang, and K.-L. Tan, "Hiding data accesses in steganographic file system" in Proc. ICDE'04, 2004. Article (CrossRef Link)
[28] C. Troncoso, C. Diaz, O. Dunkelman, and B. Preneel, "Traffic analysis attacks on a continuously-observable steganographic file system" in Proc. Information Hiding, 2007. Article (CrossRef Link)
[29] C. Diaz, C. Troncoso, and B. Preneel, "A framework for the analysis of mix-based steganographic file systems" in Proc. ESORICS, 2008. Article (CrossRef Link)
[30] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati, "Efficient and private access to outsourced data" in Proc. ICDCS, 2011. Article (CrossRef Link)
[31] Sean P. Meyn, and Robert L. Tweedie, "Computable bounds for geometric convergence rates of Markov chains" in The Annals of Applied Probability, Vol. 4, No. 4, 1994. Article (CrossRef Link)
[32] Jeffrey S. Rosenthal, "Convergence rate for Markov chains" in SIAM Review, Vol. 37, No. 3, 1995. Article (CrossRef Link)
[33] S. F. Jarner, and G. O. Roberts, "Polynomial convergence rates of Markov chains" in The Annals of Applied Probability, Vol. 12, No. 1, 2002. Article (CrossRef Link)
[34] K. Yang, J. Zhang, W. Zhang, and D. Qiao, "A light-weight solution to preservation of access pattern privacy in un-trusted clouds" in Proc. ESORICS, 2011. Article (CrossRef Link)



Ka Yang received his B.S. degree in Electrical Engineering in 2007 from Harbin Institute of Technology, Harbin, China. He is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at Iowa State University, Ames, Iowa. His current research includes data privacy, cloud computing and pervasive computing applications.



Jinsheng Zhang received his B.S. degree in Information Security in University of Science and Technology of China (USTC), China, in 2010. Since then, he joined Department of Computer Science at Iowa State University, Ames, Iowa. Currently, he is a graduate student pursuing his Ph.D. degree in Iowa State University. His research interests include cloud computing, cloud storage, data outsourcing, security and privacy.

Wensheng Zhang received the B.S. degree from Tongji University, Shanghai, the M.S. degree fromthe Chinese Academy of Sciences, Beijing, and the Ph.D. from Pennsylvania State University, all incomputer science and engineering. Since 2005, hehas been a faculty member with the Department of Computer Science at Iowa State University,wherehe is now an Associate Professor. His researchinterests include networks and distributed systems.

Daji Qiao received the Ph.D. degree in Electrical Engineering: Systems from the University of Michigan, Ann Arbor, in February 2004. He is currently an Associate Professor in the Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, USA. His current research interests include cyber security and cloud computing, protocol and algorithm innovation for IEEE 802.11 wireless local area networks and wireless sensor networks, and pervasive computing applications. He is a member of the IEEE and ACM.