Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Secure searching on cloud storage enhanced by homomorphic indexing

Shu Qin Ren[a,*], Benjamin Hong Meng Tan[a], Sivaraman Sundaram[a], Taining Wang[b], Yibin Ng[a], Victor Chang[c], Khin Mi Mi Aung[a]

[a] *Data Center Technologies Division, Data Storage Institute, A*STAR, Singapore 138932, Singapore*
[b] *School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417, Singapore*
[c] *School of Computing, Creative Technologies and Engineering, Leeds Beckett University, Leeds LS6 3QR, UK,*

HIGHLIGHTS

- A searchable encryption is presented against both data and access pattern leakage.
- A homomorphic exclusive-or (XOR) function is defined to enable the evaluation key to be calculated instead of storing.
- An effective and feasible approach performs with a query of less than 60 milliseconds among 100,000 entries.

ARTICLE INFO

Article history:
Received 29 May 2015
Received in revised form
28 January 2016
Accepted 16 March 2016
Available online xxxx

Keywords:
XOR-homomorphism encryption
Secure searching
Cloud storage

ABSTRACT

Enterprise cloud tenants would store their outsourced cloud data in encrypted form for data privacy and security. However, flexible data access functions such as data searching is usually sacrificed as a result. Thus, enterprise tenants demand secure data retrieval and computation solution from the cloud provider, which will allow them to utilize cloud services without the risks of leaking private data to outsiders and even service providers.

In this paper, we propose an exclusive-or (XOR) homomorphism encryption scheme to support secure keyword searching on encrypted data for cloud storage. First, this scheme specifies a new data protection method by encrypting the keyword and randomizing it by performing XOR operation with a random bit-string for each session to protect access pattern leakage; Secondly, the homomorphic evaluation key enables the searching evaluation to be on-demand calculated, thus it removes the dependency of key storage on cloud and enhance protection against cloud's violability; Thirdly, this scheme can effectively protect data-in-transit against passive attack such as access pattern analysis due to the randomization. This scheme also can reduce data leakage to service provider because the homomorphism-key solution instead of key storage on cloud. The above three features have been proved by the experiments and further tested out at Email service which can support secure subject searching. The execution time of one searching process is just in the order of milliseconds. We could get 2–3 times speedup compared to default utility *grep* with the concern of expensive one-time indexing which can be built off-line in advance.

1. Introduction

In contrast to traditional storage services with fully trusted infrastructure and management, cloud storage provides tenants with a transparent service, like elastic capacity and flexible accessibility, without the need to manage troublesome infrastructure. Individual users already enjoyed the flexibility, accessibility and data management provided by cloud storage services such as gmail, dropbox and wechat. People are demanding more storage space from service providers to backup [1], share documents, photos and videos with friends [2]. All these benefits are based on the

* Corresponding author.
E-mail addresses: REN_Shuqin@dsi.a-star.edu.sg (S.Q. Ren), benjamin.tan.hm@gmail.com (B.H.M. Tan), sivaraman_s@dsi.a-star.edu.sg (S. Sundaram), tainingw@gmail.com (T. Wang), NG_Yibin@dsi.a-star.edu.sg (Y. Ng), V.I.Chang@leedsbeckett.ac.uk (V. Chang), mi_mi_aung@dsi.a-star.edu.sg (K.M.M. Aung).

http://dx.doi.org/10.1016/j.future.2016.03.013
0167-739X/© 2016 Elsevier B.V. All rights reserved.

assumptions that individual users trust the service providers or take the risk to leak privacy to service providers.

For enterprise tenants/users, data growth is tremendous with online business transactions, and it will continue to be so. The demand for outsourcing data storage and management has increased dramatically. The study from TheInfoPro's Wave shows that on-premises private cloud will host 30% and off-premises public cloud will host 15% IT service by 2015 and data protection and response on data breach are two top issues to concern. However, the above assumption is not applicable to enterprise tenants as individual users. Business data is vital to companies, they can get reliable, available, fault-tolerance and performance from cloud service providers, but they cannot take the risk to let service provider to scan data. Enterprise tenants have to consider data confidentiality and safeguard the data from unauthorized access and analysis, even browsing from service provider. If enterprise tenants were willing to outsource data storage to cloud, they would prefer flexible but secure data storage and management services. For example, they would like cloud service handle their queries without knowing what is queried. Thus, effective and secure data storage and retrieval to and from cloud storage platforms become very important functions desired by cloud tenants and studied by researchers.

As data itself and its access pattern are two major aspects of privacy issues concerned by cloud tenants, the outsourced data on the cloud must leak as little information as possible. We consider the application scenario where a group of users share data through an untrusted data storage server as what they did through local network. The server can store and search on encrypted data it hosting for this group of users. It uses indexed and encrypted keywords to serve searching on encrypted data. We also require the search can be executed without leaking access pattern, such as the frequency of a query have to be protected against man-in-the-middle hacker, advantages for storage server to analyze and so on.

We are working on a searchable encryption scheme to protect data confidentiality and eliminate the access pattern leakage issue, especially to protect against passive attack from eavesdropping, but not eliminate ciphertext analysis from cloud server. We construct a new searchable encryption where the query is encrypted and randomized for each session by a XOR-homomorphic operation. We outline the contributions of this paper as the following:

- A new searchable encryption is presented that avoids both data and access pattern leakage. The exclusive-or (XOR)-homomorphic permutation function $f(.)$ is used to preserve special characteristics in the data structure after randomization so that we can conduct searches revealing neither data nor access patterns.
- A new homomorphic function $Hf(.)$ is used to enable the permutation key to be calculated during evaluation instead of key storage on cloud. This is a step of improvement against cloud's violability.
- We prove that the proposed XOR-homomorphism encryption satisfies a stronger security guarantee than existing searchable encryption scheme, such as Song's scheme. This XOR-homomorphism encryption eliminates the risks of search pattern leakage to eavesdroppers.
- We test and compare the performance of the proposed construction with existing searchable encryption schemes. The experiment results indicate the effectiveness and feasibility of proposed construction.

The rest of this paper is organized as follows. Section 2 lists some related work, Section 3 illustrates the setting and potential threats and Section 4 defines the symbols and gives backgrounds. Section 5 describes preliminary knowledge related to secure searching and random permutation. Section 6 gives a full description of our new searchable encryption scheme. Section 7 shows the experiment results and analysis. Finally Section 8 provides our conclusions.

2. Related work

Generally, searching solutions over encrypted data involve building an encrypted searchable index such that its content is hidden from the remote server yet allowing the corresponding documents to be searched. These solutions differ from each other mostly in terms of single/Multiple keyword search, types of techniques to build trapdoor functions, and sorting support for query results. Indexing is the most important for searching, from which Boolean operation can be added for multi-keyword searching and similarity score can be added for sorting.

Related work on searchable encryption scheme. Searchable encryption scheme is the key technology to enable private equality test on encrypted keywords. Song et al. proposed the most efficient searchable encryption scheme by embedding a data structure trapdoor function to test if a ciphertext keyword is in the test list [3]. Boneh used bloom filters to build trapdoor functions to evaluate whether a match occurs [4]. Chang used dictionaries to develop index schemes for searching on encrypted data [5]. Goh described an efficient secure index construction Z-IDX using pseudo-random functions and Bloom filters [6]. Some researcher extended indexing by adding private similarity comparison. A few of them, most notably [7,8], allow similarity search. These schemes extended the searching enabled index with trapdoor functions and linked similarity score to the document to form a 3-tuple of $<$ Keyword, DocID, score $>$. The new added score is also encrypted and allows secure comparison for sorting the query result set. From the above searchable encryption schemes, recent researchers extended the indexing construction for multi-keyword and sorted searching over encrypted data. Works such as [9–11] proposed multi-keyword similarity search over encrypted data. All of these schemes are efficient and provably secure in terms of confidentiality, but not in access pattern protection.

Regarding to data access pattern, groups of researcher proposed various solutions to address access pattern leaks when searching on encrypted data [12–15]. Some models like Oblivious RAM [14,13,12], do not leak any information, but it is too computationally expensive to be adopted on large datasets due to its whole dataset update. Recently some groups are working on a group-based construction to eliminate access pattern leakage [15], which tradeoffs the data privacy at less computation cost. Another groups are working on secure multi-party computation such as [16], which has expensive communication overhead. Some recent work addressed both data and access pattern privacy, they proposed more comprehensive solution to protect data stored on server, queries sent by client, and access pattern as well [17,18]. But these works have not been deployed or tested to prove their practical usage yet.

This paper is addressing both data and access pattern confidentiality with a new searchable encryption scheme, but we do not concern sorting of the query result in this paper. Multi-keyword searching can be easily extended by adding Boolean operation. We adopt the principle of Song's scheme [3] for it is practicable and provably secure in terms of data confidentiality. We further enforce protection on access pattern by introducing randomized session query and a homomorphic evaluation key to effectively protect access frequency against eavesdropper. Our scheme is implemented and tested out for its accuracy and efficiency measurement.

3. Threat model

The scenario is where group of users are sharing untrusted storage from third party. We assume data confidentiality is data owners' responsibility because only they can classify what data is sensitive. Clients encrypt their data while the untrusted server still can provide searching capability on encrypted data. The server's availability, reliability, fault-tolerance and performance can be trusted but it may be curious about data and data access pattern.

3.1. Entities

Two entities are in our system:

- Users: Legitimate users are able to access encrypted data residing on untrusted server. For data confidentiality, users encrypt data and keywords before loading to the untrusted server.
- Server: Store and retrieve encrypted data with legitimate users requests. Server provides searching ability although it knows neither data value nor query.

3.2. Adversary and attacks

Curious Server as an adversary. We model the untrusted server as honest but curious. The server is hosting the encrypted data and has full access, it may be curious about what data and what query a user is accessing.

Eavesdropper as an adversary. Eavesdropper can passively listen to the traffic and try to guess high frequency accessed data. He may further guess the query from query frequency distribution of domain knowledge.

4. Background and definitions

Our scheme uses several fundamental primitives from classical cryptography. And we adopt the standard definition of security from the provable security literature [19] and measure the cryptographic primitives in terms of the resources needed to break them as defined in Song's scheme [3].

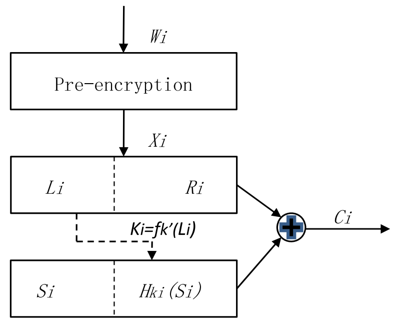
Here we list the primitives used in our systems.

1. *The distinguishing probability of an algorithm*. Let an arbitrary algorithm $A : \{0, 1\}^n \to \{0, 1\}$ and let X and Y be random variables distributed on $\{0, 1\}^n$. The distinguishing probability of A – sometimes called the advantage of A – for X and Y is

   $$Adv_A = |Pr[A(X) = 1] - Pr[A(Y) = 1]|.$$
2. *Pseudorandom generator G with (t, e)-security*. Let $G : K_G \to S$ is a (t, e)-secure pseudorandom generator if every algorithm A with running time at most t has advantage $Adv_{AG} \le \varepsilon$, illustrated as $Adv_{AG} = |Pr[A(G(U_{K_G})) = 1] - Pr[A(G(U_S)) = 1]|$, where $Adv_A \le \varepsilon$.
3. *Pseudorandom function F*. Let a pseudorandom function $F : K_F \times X \to Y$ is a (t, q, e)-secure pseudorandom function if every oracle algorithm A making at most q oracle queries and with running time at most t hash advantage Adv_{AF} less than a small value ε $Adv_{AF} = \|Pr[A^{F_k} = 1] - Pr[A^R = 1]\| \le \varepsilon$, where R represents a random function selected uniformly from the set of all maps of $X \to Y$.
4. *Pseudorandom permutation P*. If the above function F is permutation function P. Let $P : K_P \times Z \to Z$ is a (t, q, e)-secure pseudorandom function, where permutation function has inverse, P with P^{-1} and the random selected function is a random permutation selected uniformly from the set of all permutation bijections on Z, symboled as π with its inverse π^{-1}. Thus for a (t, q, e)-secure pseudorandom function, its advantage is formulated as:

   $$Adv_{AP} = \|Pr[A^{P_k, P_k^{-1}} = 1] - Pr[A^{\pi, \pi^{-1}} = 1]\| \le \varepsilon.$$

A cryptosystem is not considered secure in terms of indistinguishability if an adversary has an 'advantage' in distinguishing the chosen ciphertext, meaning that the adversary has a probability greater than the small value ε. We rely on deterministic encryption, secure permutation functions to build up our scheme to protect both data and access pattern leakages. And its security is measured according to the above definition. The notations used in our paper are listed as Table 1.



$$C_i = X_i \oplus (S_i \mid H_{ki}(S_i))$$
$$C_i \oplus X_i = (S_i \mid H_{ki}(S_i))$$

Fig. 1. Principle of secure searching scheme.

5. Preliminary works

5.1. Problems defined for search on encrypted data

Search on encrypted data without comprising data privacy means data itself and its access patterns should be protected from attackers and service providers. To deal with data leakage, both queries and results have to be protected in an encrypted form while allowing searchable functionality on the encrypted data. To deal with access pattern leakage, the encrypted data should not be distinguishable from each other and its access frequency should not be eavesdropped.

As an efficient searchable encryption, Song's scheme gave the principle of private keyword search, as diagrammed in Fig. 1. The ith keyword W_i is encrypted using a deterministic encryption, as pre-encryption result, and split into two parts left L_i and right R_i. Then this pre-encryption result is randomized by a XOR with a masking string. The masking string is formed as follows: (1) Setting the left part of the string as a random number S_i; (2) set right part of the string as the result of a hash function of the left part, denoted as H_{Ki} (S_i) with hash key Ki which is generated from a pre-defined function $f(.)$ with the input of the left part of deterministic encryption L_i; (3) form the masking string $S_i|H_{Ki}(S_i)$. Finally, the XOR of the deterministically encrypted value with the masking string is computed; it is the result denoted as C_i in Fig. 1. If same word going through the pre-encryption with the output of $X_i = L_i \mid R_i$, the XOR result of X_i and C_i will keep the structure of $S_i \mid H_{Ki}$ (S_i) as left and right part respectively, match found, not otherwise.

By evaluating the encrypted data structure between the left and right parts, Song's searchable encryption can be used for searching. The XOR operation makes this scheme practical and feasible. However, this scheme did not protect search pattern. Because each query session with same keyword generates same ciphertext, how frequent a query is searched and whether this query is different from previous one are visible to the man-in-the-middle due to the equality of query ciphertext. We adopted its practical searching principle and enhanced its security against pattern leakage issue by embedding a XOR-homomorphism permutation to randomize session query.

We reviewed Song's Theorem 4.3, as below. Each sequence $T_i = \langle S_i, F_{Ki}(S_i)\rangle$ is the random string whose first $n - m$ bit is random number S_i and last m-bit is the permutation of S_i using key Ki. The theorem shows the scheme's provable secrecy. For the random generator, given any block cipher, we may build a pseudorandom generator using the counter mode [20] or a pseudorandom function using the CBC-MAC [21] which are mentioned by Song. We use Fisher–Yates Shuffle [22] to build our permutation function $Hf(.)$ with homomorphic key. Our scheme can provide provable security with Fisher–Yates Shuffle and secure on-demand permutation key on untrusted server.

Table 1
Notations for secure searching enhanced by homomorphic evaluation key.

Notation	Description
W_i	The *i*th key word in plaintext
$X_i = E(W_i)$	The deterministic encryption of the *i*th key word
L_i	The left part/first $(n - m)$ bits of the *i*th encrypted key word
R_i	The right part/last *m* bits of the *i*th encrypted key word
$H_{Ki}(S_i)$	The hash of a random number S_i with hash key *Ki* in Song's scheme
$f_{k_{im}}(S_{im})$	Pseudo random permutation function on random number S_{im} with permutation key k_{im} in our scheme
K_i	The key used for pseudo random function $H(.)$ in Song's scheme, which is generated by function $G(.)$;
k_{im}	Permutation key at the *m*th session for the *i*th key word for permutation function $f(.)$ in our scheme, which is generated by $Hf(.)$;
S_{im}	Pseudorandom number generated at the *m*th session for the *i*th key word
C_{imL}	The left part/first $(n - m)$ bits of the *i*th encrypted and randomized key word
C_{imR}	The right part/last *m* bits of the *i*th encrypted and randomized key word

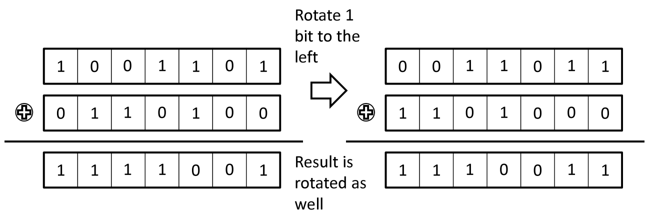


Fig. 2. Example of XOR-homomorphism function—1-bit left circular shift.

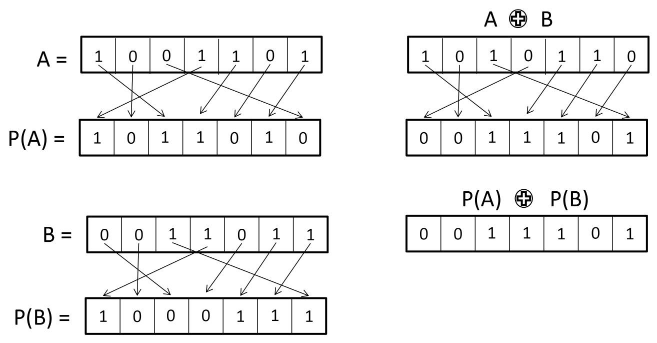


Fig. 3. Example of XOR-homomorphism function—bit-permutation.

Theorem 4.3. *Suppose E is a (t, l, e_E)-secure pseudorandom permutation, F is a (t, l, e_F)-secure pseudorandom function, f is a (t, l, e_f)-secure pseudorandom functions, G is a (t, e_G)-secure pseudorandom generator, and if the key material is chosen as described above,the random string generated is $(t - \varepsilon, e)$-secure pseudorandom generator, where ε is negligible compared to t. Then the algorithm described above for generating the sequence $\langle T_1, T_2, \dots, T_l\rangle$ will be a $(t - \varepsilon, e_H)$-secure pseudorandom generator* [3].
Note*: E and F are denoted as Pre-encryption and H in this paper, as denoted in Fig. 1 as well.*

5.2. XOR-homomorphism functions

We use pseudorandom bit-permutations to build XOR-homomorphism function to enhance the protection against data access pattern leakage issue in this paper, as detailed in Section 6. Properties of XOR-homomorphism functions are introduced here for understanding.

XOR-homomorphism refers to the property of some functions; they preserve the structure of the XOR ($\bigoplus$) operation on the input set. For a XOR-homomorphism function f and inputs x_1 and x_2, $f(x_1 \bigoplus x_2) = f(x_1) \bigoplus f(x_2)$; the XOR of the inputs when passed through the function will be equal to the XOR of the output of the function on the inputs. For example, bit-based circular shift is an XOR-homomorphism function; the result of two strings of *m* bit left circular shift followed by an XOR operation is equal to the result of an XOR operation on the strings followed by a *m*-bit left circular shift, an example of 1-bit left circular shift is illustrated in Fig. 2.

The other example of a XOR-homomorphism function is permutation on bits of data. To ensure that the randomization cannot be tampered with easily, we must be careful about choosing a permutation. The bit-permutation can be chosen with an algorithm, such as Knuth shuffle, which will result in a random bit-permutation. Fig. 3 illustrates a XOR-homomorphism of bit-permutations. The randomness of the permutation function depends on the underlying source of randomness.

As the base principle of searchable encryption showed, a special characteristic of the data is kept for the relationship between the left and right part. And if this characteristic can still be preserved after randomization, we can search without revealing anything. We add a session randomization by generating a cipher pad for each session. Each cipher pad has two parts, left and right. For each session, we use a pseudorandom number generator to generate the left half (S_i) and perform XOR-homomorphic bit permutation on the left half to generate the right half using Knuth shuffle, details can be referred 5.3.

5.3. Fisher–Yates Shuffle

As permutation has the property of XOR-homomorphism, a secure permutation could be used to assist secure searching. The question is what kind of permutation would be security. Knuth Shuffle is algorithm for generating a random permutation of a finite set by randomly shuffling [22–24]. One of the variants of Fisher–Yates shuffle is the famous Sattolo's algorithm that generates uniformly distributed cycles of length *n*. This algorithm is known for its unbiased shuffling and efficient running time.

- **Unbiased**. Permutations generated from Sattolo's algorithm fall in a uniform distribution.
- **Efficient**. The shuffle time is proportional to the number of items being shuffled and it can be customized.

6. Homomorphic exclusive-or operation enhance secure searching on cloud storage

6.1. Secure permutation with Fisher–Yates Shuffle

As shown in Fig. 4, the cipher pad we defined are based on bit permutation with Sattolo's algorithm. The Sattolo's algorithm, as introduced in Section 5.3, gives a evenly distributed random permutation which allows a uniform distributed permutation happens at each session, thus it enables session protection. More important, this algorithm is XOR-homomorphic to make it suitable for the secure searching. Thus we use the concept of Sattolo's algorithm for our permutation function. Based on Sattolo's permutation, an example of the randomize permutation pad and its XOR-homomorphic properties are illustrated as figure.

6.2. XOR-homomorphic Structure of Cipher Pad

XOR-homomorphic Structure of Cipher Pad is defined as following four steps:

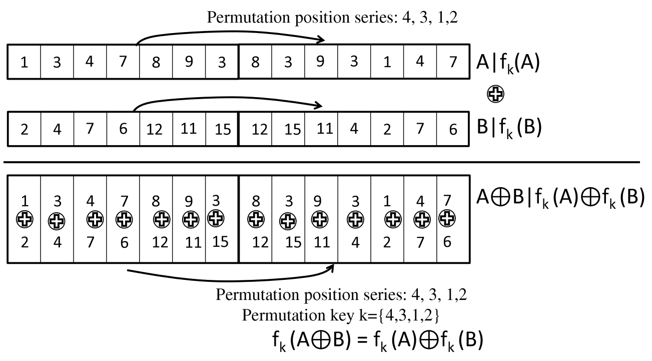


Fig. 4. Secure permutation with Fisher–Yates Shuffle to generate random pad.

1. For the ith key word W_i, it is pre-encrypted using a deterministic encryption as $X_i = L_i|R_i$ as shown in the left part of Fig. 5;
2. A random number is generated as S_{io} and it is XORed with L_i to form the current session (the oth session) permutation key $k_{io} = Hf(L_i \bigoplus S_{io})$;
3. The permutation of this random number S_{io} using the permutation key k_{io} to generate $f_{k_{io}}(S_{io})$;
4. Form the masked ciphertext for the ith key word in the oth session as: $L_i \bigoplus S_{io}|R_i \bigoplus f_{k_{io}}(S_{io})$.

Suppose this oth session ciphertext is stored at cloud as $L_i \bigoplus S_{io}|R_i \bigoplus f_{k_{io}}(S_{io})$. When the qth query session occurs, the following similar four steps are proceeded:

1. For the ith key word W_i, it is pre-encrypted using a deterministic encryption as $L_i|R_i$, as shown in the right part of Fig. 5;
2. A session random number is generated as S_{iq} and it is XORed with L_i to form the current session (the qth session) permutation key $k_{iq} = Hf(L_i \bigoplus S_{iq})$;
3. The permutation of this random number S_{iq} using the permutation key k_{iq} to generate $f_{k_{iq}}(S_{iq})$;
4. Form the masked ciphertext for the ith key word in the qth session as: $L_i \bigoplus S_{iq}|R_i \bigoplus f_{k_{iq}}(S_{iq})$;
5. The masked ciphertext is sent to server at cloud for secure searching.

As illustrated in Fig. 5, the masked cipher from the qth session is different from oth session, and the permutation key is also different with k_{io} and k_{iq} respectively. Yet the ExclusiveOR operation of these two ciphertexts generates $S_{io} \bigoplus S_{iq}|f_{k_{io}}(S_{io}) \bigoplus f_{k_{iq}}(S_{iq})$. If we set the final permutation key as $k_{io} \bigoplus k_{iq}$, $f_{k_{io}}(S_{io}) \bigoplus f_{k_{iq}}(S_{iq}) = f_{k_{io} \bigoplus k_{iq}}(S_{io} \bigoplus S_{iq})$, which still keeps the relationship between left hand and right hand and allows this scheme keep searching capability while addon computable permutation key. This new feature completely eliminates the key storage dependency on cloud storage if the server can compute k_{io} and k_{iq}. We set a XOR homomorphism function $Hf(.)$ which enables the final permutation key as $k_{io} \bigoplus k_{iq}$ with $k_{io} = Hf(C_{ioL})$, $k_{iq} = Hf(C_{iqL})$. In such a manner, only this homomorphism function $Hf(.)$ is needed to share with server. In our algorithm, we define $Hf(.)$ as the left part of the ciphertext itself, as the following equations:

$$Hf(C_{ioL}) = L_i \bigoplus S_{io} \tag{1}$$

$$C_{ioR} = R_i \bigoplus L_i \bigoplus S_{io}. \tag{2}$$

Observing the ciphertext with C_{ioL} and C_{ioR}, their XOR will generate R_i. We use R_i to build hash table to organize the encrypted keywords to speedup the searching operation by reducing its complexity close to $O(1)$ from $O(n^2)$.

6.3. Secure data upload to build secure dictionary

Two types of data to be protected on the untrusted servers before data is uploaded to it: (1) key words protection using deterministic symmetric pre-encryption and randomization; (2) data file using encryption algorithm such as AES. In this paper, keywords can be used as index to the file in which keywords are included. We will focus on the keywords protection and searching on these keywords.

Symmetric encryption is used to secure the privacy of the data and randomization is used to prevent statistical analysis by an eavesdropper on the encrypted data. The keyword to be searched is first encrypted with a deterministic symmetric encryption scheme such as block ciphers in ECB mode or with a constant initialization vector, which ensures that the encrypted data is consistent. The result of deterministic encryption is further randomized by XORing with a randomized pad as defined in 6.1 to protect the privacy of the key words.

As mentioned in the previous section, the randomized pad is created by the following procedure: (1) Generating a random number as the left part of a randomized pad; (2) Hashing the left half of the encrypted keyword as permutation key; (3) Using them to perform a pseudorandom bit-permutation as the right part of the randomized pad.

This random pad and encrypted data is XOR-ed together and generates a randomized output. This prevents any eavesdropper or man-in-the-middle from getting any information about the keyword by analyzing the traffic. For new keyword entry, the keyword goes through the symmetric encryption followed by a randomization with a random pad, and the result as a protected entry is sent to the untrusted server where they can be stored as secure index in secure dictionary or tree-based structures for further searching, as illustrated in Fig. 6.

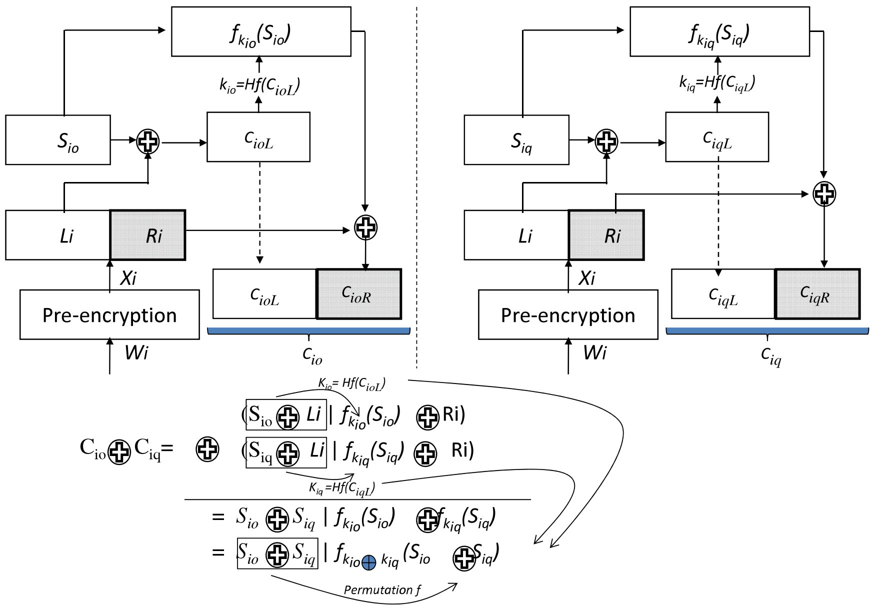
6.4. Secure data searching

The randomized and encrypted keyword C_{iq}, as illustrated in the 5-steps of Section 6.2, is sent to the untrusted server for searching, as diagrammed in Fig. 7. The server will check each entry of the secure dictionary and session query. It first calculate the permutation key with the homomorphism function $Hf(.)$ on the query and the checking entry; and then it XOR the query and entry to check whether the relationship defined by $P(.)$ between the result left partial and right partial sustain or not with the permutation key computed, as the bottom of Fig. 7 illustrated. If the relationship between left part and right part preserves as if the relationship between left part and right part preserves as $f_{k_{io}}(S_{io}) \bigoplus f_{k_{iq}}(S_{iq}) = f_{k_{io} \bigoplus k_{iq}}(S_{io} \bigoplus S_{iq})$, the entry hit and its corresponding data is the query result and will be sent to the tenant.

6.5. Secure download

If there is a entry hit, the corresponding data pointed from the entry is returned as a ciphertext. With the random number used to randomize the ciphertext stored, the trusted client can decrypt the message and retrieve the original data.

If the hitted entry or encrypted keywords are also needed to decrypt, the following procedure is executed. With random generator and seed, the random number of secure entry S_{io} is generated, the left part of the ciphertext C_{ioL} is XOR-ed with S_{io} to obtain the left part of the pre-encryption L_i. $k_{io} = Hf(C_{ioL})$ to get permutation key and $f_{k_{io}}(S_{io})$ to get the permutation pad. Form the randomized pad as $S_{io}|f_{k_{io}}(S_{io})$ then we XOR the retrieved ciphertext with the randomized pad to obtain the pre-encrypted keyword. This pre-encrypted keyword is decrypted to yield the original private data finally.



$f_{kio}(S_{io})$
$k_{io}=Hf(C_{ioL})$
S_{io}
C_{ioL}
Li
Ri
Xi
Pre-encryption
Wi
C_{ioL}
C_{ioR}
C_{io}
$f_{kiq}(S_{iq})$
$k_{iq}=Hf(C_{iqL})$
S_{iq}
C_{iqL}
C_{iqR}
C_{iq}
$K_{io}=Hf(C_{ioL})$
$K_{iq}=Hf(C_{iqL})$
$C_{io} \oplus C_{iq} = \oplus (S_{io} \oplus Li \mid f_{kio}(S_{io}) \oplus Ri)$
$(S_{iq} \oplus Li \mid f_{kiq}(S_{iq}) \oplus Ri)$
$= S_{io} \oplus S_{iq} \mid f_{kio}(S_{io}) \oplus f_{kiq}(S_{iq})$
$= S_{io} \oplus S_{iq} \mid f_{kio \oplus kiq}(S_{io} \oplus S_{iq})$
Permutation f

Fig. 5. On-demand key computing with XOR-homomorphism.

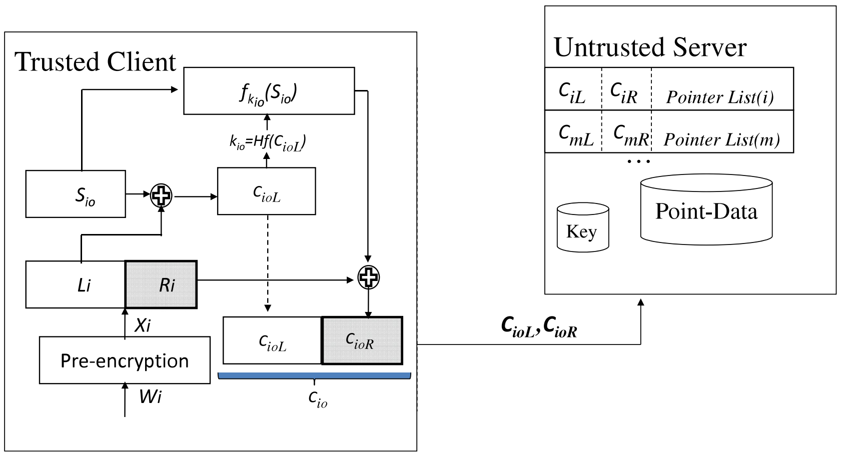


Fig. 6. Secure data upload to build secure dictionary.

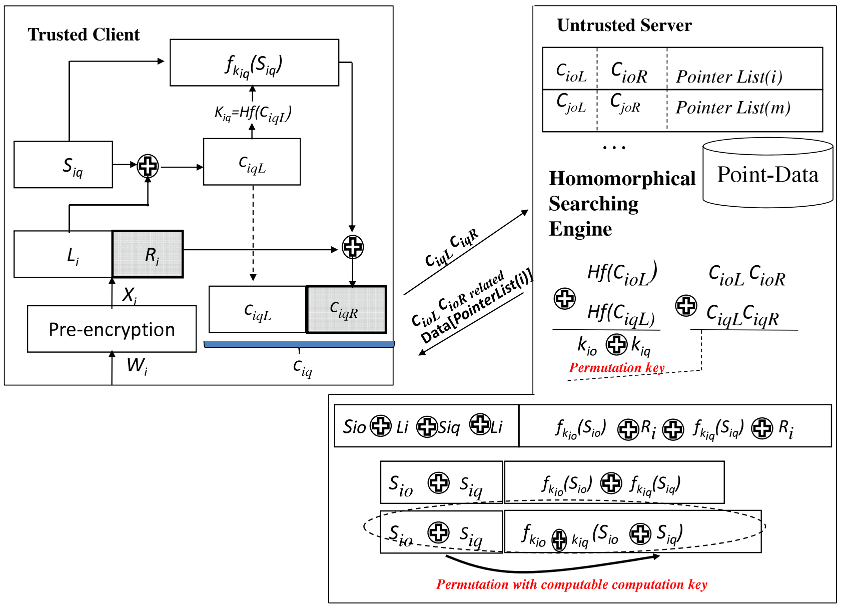
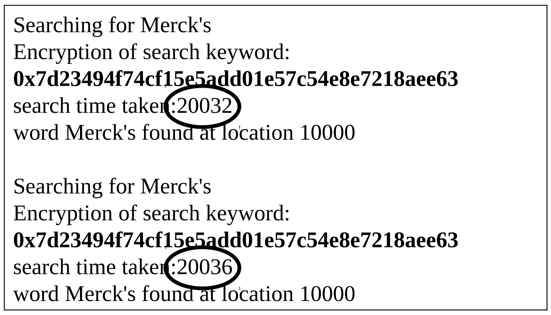


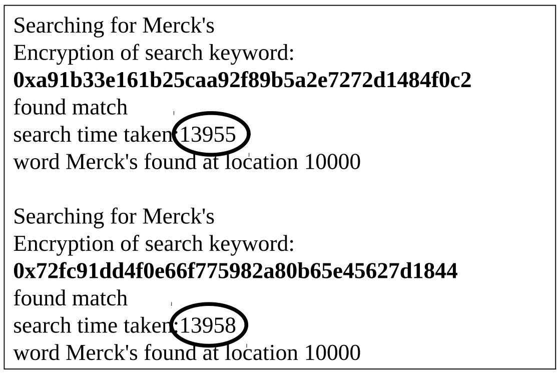
Fig. 7. Secure data searching.



```
Searching for Merck's
Encryption of search keyword:
0x7d23494f74cf15e5add01e57c54e8e7218aee63
search time taken:20032
word Merck's found at location 10000

Searching for Merck's
Encryption of search keyword:
0x7d23494f74cf15e5add01e57c54e8e7218aee63
search time taken:20036
word Merck's found at location 10000
```

Fig. 8. Without session protection of song's secure searching scheme.



```
Searching for Merck's
Encryption of search keyword:
0xa91b33e161b25caa92f89b5a2e7272d1484f0c2
found match
search time taken:13955
word Merck's found at location 10000

Searching for Merck's
Encryption of search keyword:
0x72fc91dd4f0e66f775982a80b65e45627d1844
found match
search time taken:13958
word Merck's found at location 10000
```

Fig. 9. With session protection of our searching scheme.

7. Experiment results and analysis

7.1. Performance test on keywords searching

We performed some experiments to verify the correctness of our scheme and compare the performance overhead with the practical scheme, Song's scheme [3], which just uses the deterministic encryption for query protection. For same query, it shoots same secure query at different sessions. Fig. 8 gives one of the results from Song's scheme. This leaves it vulnerable to ciphertext analysis and access pattern analysis attacks. We enhance the security by introducing session protection—the same keywords result in different ciphertexts in different sessions; as seen from the bolded strings in Fig. 9. Due to the session randomness from the pseudorandom permutation, it effectively protects the query from the above two attacks.

Secondly, we compare the time taken to search the data in the server. The process time includes two parts: query protection at client side and query searching at server side. With the extra session protection, our scheme adds an additional overhead of about 880–900 ms for the session protection at client side. But the searching time at the server side is much faster in our scheme compared to the existing scheme, with 13 955 vs. 20 032 ms, more than 30% speed up. Furthermore, as evidenced in Table 2, our schemes advantage increases if the keyword is deeper down the list to be searched. The additional overhead of about 880–900 ms at client side is negligible compared to the searching time taken on the server side, which is about 14 000 ms for a list of 10 000 words.

7.2. Use case on email service

Mailbox searching among encrypted emails on an untrusted server is one of the real-life applications of our private searching scheme. We have implemented the XOR-homomorphism enhanced secure searching scheme in the context of email searching, set up Postfix on Ubuntu 13.10 as the Mail Transfer Agent (MTA),

Table 2
Performance tests on the time taken to search encrypted keywords.

Found location	Searching time (ms)	
	Our scheme	Existing scheme
2500	3504/3524	4941/4947
5000	6984/6995	9913/9989
7500	10 398/10 572	14 826/14 913
10 000	13 955/13 958	20 032/20 036
Non-exist	14 032	19 769

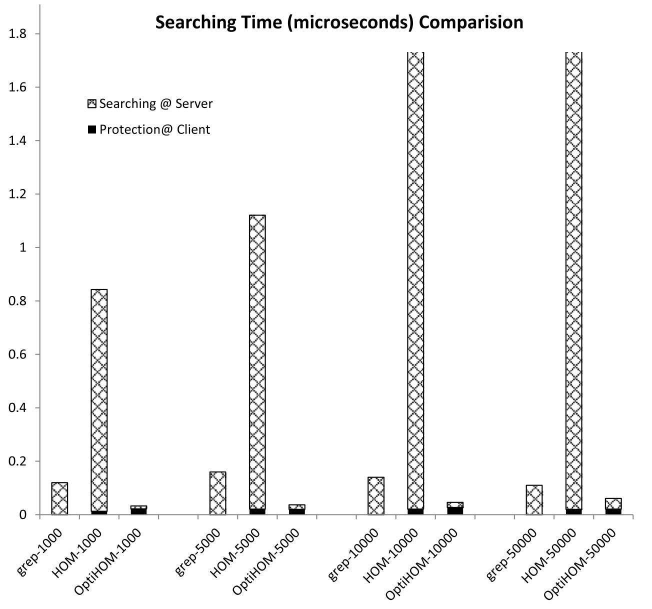


Fig. 10. Performance evaluation compared with default searching.

and performed experiments under our searching scheme. Instead of using mbox, the traditional mail storage format for Linux, we choose Maildir format, which stores each email in a separate file. For experimental purpose, only subjects of the emails are encrypted and searched in this set up.

We compare the performance of our searching scheme with the Linux built-in search utility, grep command, as shown in Table 3. Our solution has three phases: phase 1 for data protection at client side; phase 2 for loading and updating index at server side; phase 3 for searching at server side. Our native XOR-homomorphism solution is named as HOM and the optimized solution with indexing is named as OptiHOM. As the experiment results shown in table, phase 2 is an expensive process. But phase 2 could be one-time off-line process followed by incremental on-line update. For analyzing the online searching performance, only phase 1 and phase 3 are compared in our set up.

Obviously the searching time at phase 3 is expensive compared with default search of grep, ranging from 8 to 15 times slower with the increase of keyword list as shown in Table 3. In order to make our solution competitive with the default search yet provide protection, we introduce hash-based dictionary for speeding up. Each encrypted keyword and its corresponding DocID are stored as value and its index is the key. When a query is sent to server, it is attached with an extra index key, and the hash of this index key assists the server to filter out most of the entries in the keyword list, thus the XOR-homomorphism operation is only operated in a much smaller set.

We evaluate default grep, our naive solution HOM and indexed solution OptiHOM with the measurement of the searching time, as shown in table and Fig. 10. The results in Fig. 10 show that optimized solution, OptiHOM, is even faster than grep 4–7 times of the searching time regardless of the key word list (depth of

Table 3
Email service performance comparison between default grep and our search scheme.

Email #	Searching time (ms)						
	Grep	Our XOR-enhanced searching					
		Phase 1 encryption and randomization		Phase 2 loading and updating		Phase 3 searching	
		HOM	OptiHOM	HOM	OptiHOM	HOM	OptiHOM
1 000	120	13	22	41 680	24 350	830	11
5 000	160	21	20	196 720	200 430	1 100	17
10 000	140	21	27	640 550	643 930	2 420	19
50 000	110	20	21	775 790	938 060	9 250	40
100 000	130	11	25	1 073 630	1 334 640	19 710	33

Table 4
Secure searching time comparison of our naive scheme and optimized scheme.

Entries	Rounds	Searching time (s)			
		HOMsearch	OptHOMsearch	Norm HOM	Norm OptHOM
1 000	516	0.546116	0.013363	0.000546116	1.3363E−05
5 000	2 549	11.805252	0.085855	0.00236105	1.7171E−05
10 000	5 044	46.456223	0.117946	0.009210195	1.17946E−05
15 000	7 501	104.388108	0.179173	0.013916559	1.19449E−05
20 000	10 033	185.039314	0.238945	0.018443069	1.19473E−05
50 000	25 002	1173.794301	0.809676	0.046948016	1.61935E−05
75 000	37 490	2603.209873	1.007937	0.069437447	1.34392E−05
100 000	49 914	4638.139905	1.350909	0.092922625	1.35091E−05
1 000 000	501 330	**130** (**h**)	29.094767	0.468	2.90948E−05
5 000 000	2 527 875	**N.A.**	150.980269	**N.A.**	3.01961E−05

query entry). Since our solution has extra data protection overhead at client side, the on-line search performance includes session protection at client side plus searching on server side, as the results shown in Fig. 10; the on-line searching time of OptiHOM is still 2–4 times faster than grep even with big dataset of 100,000. These results give us the confidence to put OptiHOM solution to support big data search for cloud-backed applications.

7.3. More experiments on the optimized scheme

Due to the hash based dictionary, we are able to speedup the searching process by reducing the complexity close to $O(1)$ from $O(n^2)$. To investigate the benefits from this dictionary, we did more experiments with different depth of secure dictionary, as listed in Table 4. Entry # is the dictionary depth, Rounds is the number of queries sent, HOMsearch and Opt HOMsearch are the total searching time for our naive scheme and Optimized scheme; Norm HOM and Norm OptHOM are normalized searching time, means the time for each query. Table 4 shows that Norm OptHOM is quite stable, ranging 1.17$\tilde{1}$.35E−05 s regardless of the dictionary depth while Norm HOM changes tremendously according to the dictionary depth, which confirms our statement on the complexity of $O(1)$.

7.4. Accuracy and performance analysis

Accuracy analysis. Since each session query from clients is randomized, there is a slim chance that the result of the equality test returns a false positive. In this section, we analyze the probability that such a scenario occurs.

The equality test procedure computes the XOR of the ciphertext and the query and we denote this result as $L_r|R_r$, where $|L_r| = |R_r|$; this is then checked for the following relation, $R_r \overset{?}{=} f_k(L_r)$. The function $f_k(\cdot)$ is chosen in this setting as a pseudorandom permutation of bits which implies that it is a bijection. We consider a false positive query as a random string R^* such that R^* is not a correct query. Then, a false positive result happens when $R^* \oplus E(w) \oplus (L|f_k(L))$ is of the form $X|f_k(X)$, for some word w and mask $L|f_k(L)$.

Since the mask is of the form $L|f_k(L)$, the XOR of R^* and $E(w)$ must be of the same form as well. Let $L_a|R_a = R^* \oplus E(w)$, $|L_a| = |R_a| = \frac{n}{2}$. Since R^* is a random string and $E(w)$ is pseudorandom, we can assume that L_a and R_a are random strings themselves. Without loss of generality, we fix L_a and consider $f_k(L_a)$ and let $\Pr[f_k(L_a) = R_a]$ be the probability that the equality test returns a false positive. The false positive probability is $\Pr[H_k(L_a) = R_a] = 1/2^{n/2}$. This probability is negatively proportional to the length n of ciphertext.

Searching performance. As the OptiHOM solution makes use of hash-based index entry, the searching complexity comes down close to $O(1)$ from $O(n^2)$. This can be confirmed from the experiment results as listed in Table 4 Norm Opt Column, where each session searching is calculated by the total searching time (Opt Secsearch Column) divided by the rounds. Each searching takes around 11$\tilde{1}$3 µs regardless of the entry depth. This performance is enhanced by trading off some level of pattern security. We are getting such good performance by leaking right half of the deterministic encryption result.

8. Conclusions

We proposed a XOR-homomorphism encryption to enhance the data protection over searching process by introducing randomization for each session. Furthermore, this randomization based searching can compute the session key on-demand and does not require key storage on cloud as existing schemes. We also enhance the searching performance by a hashing based indexing. In terms of security, the XOR-homomorphism encryption allows session to be randomized by using on-demand bit permutation, which provides strong protection for query session against access pattern analysis attacks from eavesdropper and also reduce key leakage risk to cloud provider. In the measurement of searching time, our solution is even faster by using hash-based indexing than the default clear text utility grep, which not only speedup searching process but also protects access pattern leakage against eavesdropping. The use case with email server demonstrates that our solution is a lightweight privacy protection and is very practical for cloud backed applications.

Acknowledgment

The authors would like to thank Agency for Science, Technology and Research (A*STAR) for supporting the project under Future Data Centre Technologies grant; the project is entitled Secured Large Scale Shared Storage System (Grant No. 1122804006).

References

[1] Amazon, Amazon simple storage service. URL: http://aws.amazon.com/s3.
[2] Megaupload, Megaupload. URL: http://www.megaupload.com.
[3] D. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proc. of IEEE Symposium on Security and Privacy, 2000, p. 44.
[4] D. Boneh, G.D. Crescenzo, R. Ostrovsky, G. Persiano, Public key encryption with keyword search, in: Proc. of Proc. of EUROCRYPT 2004, 2004, pp. 506–522.
[5] Y.-C. Chang, M. Mitzenmacher, Privacy preserving keyword searches on remote encrypted data, Cryptology ePrint Archive, Report 2004/051, 2004, http://eprint.iacr.org/.
[6] E.-J. Goh, Building secure indexes for searching efficiently on encrypted compressed data, Online (2003). URL: http://eprint.iacr.org/2003/216/.
[7] M. Blanton, M.J. Atallah, K.B. Frikken, Q. Malluhi, Secure and efficient outsourcing of sequence comparisons, in: Foresti ESORICS 2012, in: LNCS, 2012, pp. 505–522.
[8] M. Blanton, Achieving full security in privacy-preserving data mining, in: Proceedings of the 3rd IEEE International Conference on Privacy, Security, Risk and Trust, 2011.
[9] M. Strizhov, I. Ray, Multi-keyword similarity search over encrypted cloud data, Cryptology ePrint Archive, Report 2015/137, 2015, URL: https://eprint.iacr.org.
[10] L. Chen, X. Sun, Z. Xia, Q. Liu, An efficient and privacy-preserving semantic multi-keyword ranked search over encrypted cloud data, Int. J. Secur. Appl. (2014).
[11] F. Baldimtsi, O. Ohrimenko, Sorting and searching behind the curtain, in: Financial Cryptography, 2015.
[12] G. Oded, O. Rafail, Software protection and simulation on oblivious rams, Vol. 43, 1996. URL: http://doi.acm.org/10.1145/233551.233553.
[13] R. Ostrovsky, Software protection and simulation on oblivious rams (mit Ph.D. thesis), 2009, Online. URL: http://www.cs.ucla.edu/~rafail/PUBLIC/09.pdf.
[14] B. Pinkas, T. Reinman, Oblivious ram revisited, Online (2010). URL: http://eprint.iacr.org/2010/366.pdf.
[15] C. Liu, L. Zhu, M. Wang, Y. Tan, Search pattern leakage in searchable encryption: Attacks and new construction, Inform. Sci. 265 (2014) 176–188.
[16] R. Canetti, Studiesin secure multi-party computation and applications (Ph.D. thesis), Weizmann Institue of Science, Israel, 1995.
[17] P. Vasilis, K. Fernando, V. Binh, K. Vladimir, M. Tal, C.S. Geol, G. Wesley, K. Angelos, B. Steve, Blind seer: A scalable private dbms, in: Proceedings of the 2014 IEEE Symposium on Security and Privacy, SP'14, IEEE Computer Society, Washington, DC, USA, 2014, pp. 359–374. http://dx.doi.org/10.1109/SP.2014.30.
[18] B.A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, S.M. Bellovin, Malicious-client security in Blind Seer: A scalable private DBMS, in: IEEE Symposium on Security and Privacy, 2015. URL: http://www.ieee-security.org/TC/SP2015/papers/6949a395.pdf.
[19] M. Bellare, Practice-oriented provable-security, in: Proceedings of First International Workshop on Information Security, (ISW 97), in: Lecture Notes in Computer Science, vol. 1396, Springer-Verlag, 1998.
[20] M. Bellare, A. Desai, E. Jokipii, P. Rogaway, A concrete security treatment of symmetric encryption: Analysis of the des modes of operation, in: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, IEEE, 1997.
[21] M. Bellare, J. Kilian, P. Rogaway, The security of the cipher block chaining message authentication code, in: Proceedings of CRYPTO'94, in: Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994.
[22] A.F. Ronald, S.Y. Frank, Statistical Tables for Biological, Agricultural and Medical Research, third ed., Oliver & Boyd, London, 1948.
[23] E.K. Donald, Seminumerical algorithms, Art Comput. Program. 2 (1969) 124–125.
[24] R. Durstenfeld, Algorithm 235: Random permutation, Commun. ACM 7 (1964) 420–421.

Shu Qin Ren received her Ph.D. and bachelor degrees of Computer Engineering from Korea Aerospace University in 2009 and Hefei University of Technology in 1997 respectively. She is currently a scientist with A*STAR, Data Storage Institute, Singapore. Her research interests include data privacy and security, secure data sharing and computing on cloud storage, and distributed QoS storage.

Benjamin Hong Meng Tan received the B.Sc. (Hons) degree in Mathematical Sciences from Nanyang Technological University (NTU) in 2012. He is currently pursuing a Ph.D. at the School of Physical and Mathematical Sciences of NTU. His research interests include aspects of fully homomorphic cryptography, including fully homomorphic encryption and multilinear maps.

Sivaraman Sundaram received his B.E. in Computer Science Engineering (1st Class Honors) from Anna University Chennai and M.Sc. degree from Nanyang Technological University, Singapore. He is currently a research engineer in Data Storage Institute, Singapore. His research interests include Storage and Network Security.

Taining Wang is pursuing Bachelor of Computing (Honours) in the Department of Computer Science at National University of Singapore. She is expected to receive her Bachelor's degree in 2016. She did industry attachment at Data Storage Institute (Agency for Science, Technology and Research (A*STAR)) in Singapore in summer of 2014. She also visited Ecole Polytechnique in France, and worked with the AMIB team in LIX—INRIA Saclay in summer of 2015. Her research interests include Data Storage, Query Processing, Data Encryption, and Algorithms.

Yibin Ng received the B.Eng. degree from the National University of Singapore (NUS), in 2006 and the Ph.D. degree from Carnegie Mellon University (CMU), USA, in 2012, both in electrical engineering. He is currently a Scientist in Data Storage Institute (DSI), Agency for Science, Technology and Research (A*STAR), Singapore. His research interests include security, cryptography, signal processing, error correction codes, data storage.

Chang Victor is a Senior Lecturer in the School of Computing, Creative Technologies at Leeds Beckett University, UK and a visiting Researcher at the University of Southampton, UK. He is an expert on Cloud Computing and Big Data in both academia and industry with extensive experience in related areas since 1998. Dr. Chang completed a PGCert (Higher Education) and Ph.D. (Computer Science) within four years while working full-time. He is the founding chair of international workshops on Emerging Software as a Service and Analytics and Enterprise Security. He is a joint Editor-in-Chief (EIC) in International Journal of Organizational and Collective Intelligence and a founding EIC in Open Journal of Big Data. He is the Editor of a highly prestigious journal, Future Generation Computer Systems (FGCS).

Khin Mi Mi Aung received a Ph.D. degree of Computer Engineering from Korea Aerospace University, in 2006. She is currently a scientist with A*STAR, Data Storage Institute, Singapore. Her research interests include data and information security, data center and network storage technologies.