IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON SMART CACHING, COMMUNICATIONS, COMPUTING AND CYBERSECURITY FOR INFORMATION-CENTRIC INTERNET OF THINGS

Received May 23, 2018, accepted June 21, 2018, date of publication July 4, 2018, date of current version July 30, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2852329

M-SSE: An Effective Searchable Symmetric Encryption With Enhanced Security for Mobile Devices

CHONGZHI GAO[1,2], SIYI LV[3], YU WEI[3], ZHI WANG[3], ZHELI LIU[3], AND XIAOCHUN CHENG[4], (Senior Member, IEEE)

[1]School of Computer Science, Guangzhou University, Guangzhou 510006, China
[2]State Key Laboratory of Cryptology, Beijing 100878, China
[3]College of Computer and Control Engineering, Nankai University, Tianjin 300071, China
[4]Department of Computer Science, Middlesex University, London NW4 4BT, U.K.

Corresponding author: Zhi Wang (zwang@nankai.edu.cn)

This work was supported in part by the Natural Science Foundation of Guangdong Province for Distinguished Young Scholars under Grant 2014A030306020, in part by the Guangzhou Scholars Project for universities of Guangzhou under Grant 1201561613, in part by the Science and Technology Planning Project of Guangdong Province, China, under Grant 2015B010129015, and in part by the State Key Laboratory of Cryptology, Beijing, China.

ABSTRACT Searchable encryption allows mobile devices with limited computing and storage resources to outsource data to an untrusted cloud server. Users are able to search and retrieve the outsourced data; however, it suffers from information and privacy leakage. The reason is that most of the previous works rely on the single cloud model, which allows that the cloud server gets all the search information from users. In this paper, we present a new scheme M-SSE that achieves both forward and backward security based on a multi-cloud technique. The new scheme is secure against both adaptive file injection attack and size pattern attack by utilizing multiple cloud servers. Experiment results show that our scheme is effective compared with the other existing schemes.

INDEX TERMS Searchable symmetric encryption, multi-cloud technique, forward security, backward security.

I. INTRODUCTION

With the wide use of cloud computing, a huge amount of data are outsourced to the cloud servers from users with limited computing and storage resources in Internet of Things (IoT). Though cloud computing is able to provide the powerful outsourcing services to users, the security and privacy have become challenges. The reason is that the cloud servers and the users are not in the same trust domain. Usually, the cloud servers cannot be fully trusted by the users in the system. As a result, how to protect the security and privacy of users is a critical issue for the wide application of cloud computing [20].

In order to protect data security, the users usually encrypt their data before uploading them to the cloud [26]. Keyword search and other data utilization become challenging for applications in outsourced data, such as machine learning and other data utilization with privacy protection [11], [12]. Traditional encryption methods such as symmetric encryption or hybrid encryption can be used here to protect the data security. However, after data encryption, data operations become a challenge because the users cannot perform operation over ciphertexts such as keyword search and range query [22].

Searchable encryption is one of the most basic preliminaries for the data utilization in cloud computing [29]. However, all the previous works have only considered the basic security requirements such as the confidentiality of data and revocation of the search privilege etc. However, in cloud computing environment, the adversary would launch stronger and different attacks for the cloud data [16], [42], [44]. Thus, the security and privacy issues for searchable encryption have to be considered for variants attacks.

A. SEARCHABLE ENCRYPTION

To implement keyword search over ciphertexts, searchable encryption requires the client to upload both the

keyword ciphertexts and the encrypted documents to the cloud [6], [34]. When the client wants to search the documents containing a certain keyword, the client generates a trapdoor for this keyword and sends it to the server. After confirming that a keyword ciphertext is matched with this trapdoor, the server returns the matched document identifiers back to the client [23]. Then, the client can retrieve the document by its document identifier.

There are two kinds of searchable encryption methods, Public Key Encryption with Keyword Search (PEKS) [1] and Searchable Symmetric Encryption (SSE). Derived from public key cryptographic primitives, PEKS is mainly applied in the design of complex SE schemes, such as conjunctive-subset, multi-dimensional keyword search. Based on the symmetric primitives, SSE can achieve good performance, and it has gained more attention and been widely used in encrypted storage and encrypted email systems.

1) LEAKAGES OF SEARCHABLE ENCRYPTION

However, in SSE, the (often symmetric) deterministic encryption enables the cloud server to observe the repeated queries and other leakages [7], [19]. Typically, these leakages can be divided into three types:

- Size pattern [39]. Some leakages of the SSE scheme are about the size of the entries, the total number of keywords and so on. That is to say, the server can learn the number of keyword-document pairs stored in a database.
- Search pattern [10]. This means that the server can learn that the current query is linked with a past query. The server can also learn the deterministic tokens of repeated queries.
- Access pattern [10]. The server can observe the access operation of the client, and then it can learn the document identifiers and document matching keyword. We can store documents in an ORAM when we request the documents; the access pattern will be oblivious.

2) ATTACKS OF SEARCHABLE ENCRYPTION

By abusing these leakages, the malicious server can launch attacks such as inference attack [15], leakage abuse attack [5], file injection attack [43], and so on. For the adversary, no matter how small the leakage is, it can attack SSE schemes to reveal the client's privacy. The file injection attack is a novel attack that abuses the leakage of the access pattern. It injects files containing pre-defined keywords into the client and observes the file access pattern on the server. When an injected file is fetched but others are not, the malicious server can determine which keyword is searched in the fetched injected file. In this way, the malicious server can know the query privacy and even the plaintext of the encrypted document.

The above leakages can be protected by utilizing ORAM, like TWORAM [13]. Unfortunately, ORAM always involves a huge bandwidth cost, massive storage usage and frequent interactions between the client and the server. That is to say, the SSE schemes based on ORAM are not practical [17]. As a result, when designing a practical SSE scheme, we must solve the trade-off between efficiency and leakage. Efficiency involves storage requirements, latency and bandwidth, while leakage concerns the size pattern, search pattern and access pattern. Considering efficiency, most SSE schemes abandon the ORAM and assume these leakages can be allowed. Although some attacks have been proposed as described above, in 2016, Zhang et al. [43] noted that the adaptive file injection attack cannot be applied to a forward privacy SSE scheme. Thus, the forward privacy becomes the basic security goal of a practical SSE scheme.

B. MOTIVATION

Until now, there have only been a few schemes that achieve forward privacy [3]. If the SSE scheme supports forward privacy, it means that a malicious server cannot learn whether a newly added document matches previous search queries. Backward privacy is defined in the sense that a searching keyword does not reveal the matching documents identifiers. In 2017, Bost *et al.* [4] proposed an SSE scheme that achieves both forward and backward security. As far as we know, it is the first scheme that is not based on ORAM to achieve forward and backward security.

We stress that there is still too much information (size pattern, search pattern, access pattern) to be leaked in SSE schemes that achieve forward and backward security. And many attacks can be launched. Further reducing leakages is the objective of the design of the SSE scheme. Therefore, we want to draw support from the multi-cloud technique to reduce the leakage and improve efficiency. In other words, we try to distribute these leakages into different non-colluding clouds. For example, we allow a single cloud to observe part of the size and search information, but we do not leak the whole size pattern or search pattern to each cloud because the clouds are non-colluding.

C. OUR CONTRIBUTIONS

We construct a forward and backward searchable encryption scheme based on a multi-cloud technique called "M-SSE". This is a dynamic SSE scheme that supports both add and delete operations, and it shows optimal performance compared to typical SSE schemes. More specifically, we draw support from the multi-cloud technique to distribute part of the leakages to different clouds and avoid a single cloud knowing the whole size or search pattern. Therefore, M-SSE achieves small leakages.

As shown in Table 1, we can conclude that: 1) M-SSE and Fides have the same asymptotic complexity in search and update. The experiment result shows that M-SSE is $2\times$ greater than Fides [4] in terms of speed because the client can get the index matching keyword w from two clouds simultaneously in M-SSE; 2) M-SSE and Fides can achieve both forward and backward privacy, but only M-SSE achieves the protection of the size pattern by distributing leakages to different clouds.

TABLE 1. Comparison with prior SSE schemes. N is the number of entries, which can be written as keyword/document pairs in the database, while W is the number of distinct keywords, and D is the number of documents; n_w is the size of the search result set matching keyword w, and a_w is the total number of entries matching keyword w. FP denotes forward-private, BP denotes backward-private, and SiP denotes size pattern.

Scheme	Computation		Communication		FP	BP	SiP
	Search	Update	Search	Update			
TWORAM	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^2 N)$	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^3 N)$	$\surd$	–	$\surd$
$\Sigma o\varphi o\varsigma$ [2]	$O(a_w)$	$O(1)$	$O(n_w)$	$O(1)$	$\surd$	–	–
Fides	$O(a_w)$	$O(1)$	$O(a_w)$	$O(1)$	$\surd$	II	–
M-SSE	$O(a_w)$	$O(1)$	$O(a_w)$	$O(1)$	$\surd$	II	$\surd$

II. RELATED WORK

A. SEARCHABLE ENCRYPTION

Cloud computing allows users to share cloud data securely with other users [20] and outsource computing to the cloud servers [36], [37], [40]. How to perform the data utilization is a critical problem. In 2000, Song *et al.* [38] proposed the first practical searchable encryption scheme implemented with symmetric primitives. Many SSE schemes were subsequently proposed, including static and dynamic schemes [8], [9]. Compared to dynamic SSE scheme, the static SSE scheme does not support adding and deleting entries in the initialized database or storage system. From a practical point of view, dynamic SSE schemes are much more valuable.

Generally speaking, there are three approaches: inverted index, tree and direct index of SSE. Among them, the inverted index approach is used the most widely. The element in inverted index is a (key, value), where key is a keyword and value is the document identifiers that contain the keyword.

When searching for a keyword, we obtain all the document identifiers matching the keyword. Therefore, compared with other approaches, the inverted index approach costs the minimal search time. This approach [10] was first proposed in 2006; later, many SSE schemes [18], [35] were proposed based on it.

There are also many other works on the data utilization for cloud computing, such as the privacy machine learning [25]. To solve these above data utilization problems, some cryptographic techniques have been proposed such as homomorphic cryptography [30], [31], [41], cloud data retrieval and verifiable computing [28]. The multiple cloud setting has also been considered by [21] to realize the reliability of the data deduplication. Actually, there are many similarities between the construction of deduplication and keyword search because both of them need to perform search over the cloud data with some query.

Constructing the SSE scheme is challenging. We should achieve less leakage and more optimal search and update complexity. The SSE scheme implemented with ORAM [13], [14] achieves the highest security guarantee. However, the overhead of ORAM makes the search and update complexity less optimal.

B. FORWARD AND BACKWARD PRIVATE SE SCHEMES

The notion of ORAM [24], [33] has been proposed to protect the privacy of data retrieval. Stefanov *et al.* [39] proposed an ORAM-inspired forward-private SSE construction and was the first to outline the forward privacy concept. Later, Bost [2] proposed a forward private SSE scheme that is only implemented with trapdoor permutation and achieved optimal search and update complexity. In 2016, Zhang *et al.* [43] noted that the forward private scheme can defend against the adaptive file injection attack; thus, forward privacy has become the basic security goal of the SSE scheme. Until now, several forward private SSE schemes have been proposed, including sizepattern [39], TWORAM [13], $\Sigma o\varphi o\varsigma$ [2], and so on.

In 2017, Bost *et al.* [4] gave a formal backward private definition and proposed the first SSE scheme that achieves both forward and backward security without utilizing ORAM. According to the number of metadata leakages about the inserted and deleted entries, there are three levels of backward private definitions: 1) Weak backward privacy. When inserting documents, it leaks the document's currently matching keyword w, when the update operations occurred, and the information of the cancel operation; 2) Backward privacy with update pattern. Compared with weak backward privacy, when inserting documents, it reduces the information leakage of insertion and deletion; 3) Backward privacy with insertion pattern. Compared with Backward privacy with update pattern, when inserting documents, it further reduces information leakage when the update operations occurred.

C. TRAPDOOR PERMUTATION TECHNIQUE

A trapdoor permutation family $\prod$ comprises three algorithms: *Gen*, *Eval* and *Invert*.

- $Generate(1^\lambda)$ is a randomized generation algorithm; the input is 1^λ and the output is the description i of a permutation along with the corresponding trapdoor td and $x \xleftarrow{R} D_i$. Furthermore, td is the trapdoor allowed to evaluate π^{-1}.
- $Evaluate(1^\lambda, i, x)$ is a deterministic algorithm, which takes as input $i \leftarrow Gen(1^\lambda)$, $x \xleftarrow{R} Sample(1^\lambda, i)$ and outputs $y \in D_i$. $Eval(1^\lambda, i, \cdot)$ is a permutation od D for all $(i, td) \xleftarrow{R} Gen(1^\lambda)$.
- $Invert(1^\lambda, (i, td), y)$ is a deterministic algorithm, which takes as input $(i, td) \xleftarrow{R} Gen(1^\lambda), y \in D_i$ and outputs $x \in D_i$. Therefore, $Invert(1^\lambda, (i, td), \cdot)$ inverts a permutation.

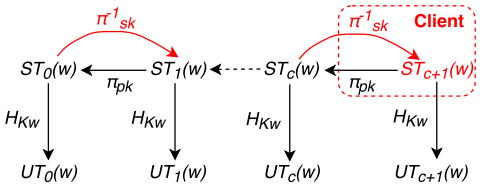


FIGURE 1. Trapdoor permutation technique.

The output of the *Generate* algorithm is a probability distribution Π on permutations; therefore, $(\pi, \pi^{-1}) \xleftarrow{R} \Pi$; here, π is a permutation and π^{-1} is the inverse permutation.

Definition 1: The advantage of algorithm A in inverting a trapdoor permutation family is:

$$Adv_{\mathrm{A}}^{Invert} = P[x = \mathrm{A}(1^{\lambda}, (i, td), y) : (i, td), \\ x \xleftarrow{R} Generate(1^{\lambda}), y \leftarrow Evaluate(1^{\lambda}, i, x)].$$

TDP in $\Sigma o\varphi o\varsigma$. As shown in Figure 1, only the client who owes the trapdoor sk can generate the permutation (forward chain), but the server who owes the public info pk can evaluate this permutation (backward chain). We can also see that the permutation about w is built by encrypting a random value $ST_0(w)$ many times using the trapdoor sk (i.e., π_{sk}^{-1}). Meanwhile, it can be retrieved by decrypting the last one using the public info pk (i.e., π_{pk}).

The $\Sigma o\varphi o\varsigma$ [2] exploits the TDP to achieve the forward secure searchable encryption. In particular, for each keyword w, there is a permutation over the set that contains the search tokens of keyword w denoted by $D(w)$. In $\Sigma o\varphi o\varsigma$ [2], the client maintains a counter c and its search token $ST_c(w)$ for each keyword w. When a keyword-document pair for keyword w is added, the client will first produce a search token by $ST_{c+1}(w) \leftarrow \pi_{sk}^{-1}(ST_c(w))$ and then produce a storage position (update token) $UT_{c+1}(w)$ using a keyed hash, and finally update its client state and store the document identifier in $UT_{c+1}(w)$ in the server. Because the latest search token is stored in the client, the malicious server cannot know where the $UT_{c+1}(w)$ is produced.

III. PRELIMINARIES

A. NOTATIONS

Let λ be the security parameter and $negl(\lambda)$ be the negligible function. As a λ bits string, the symmetric key is sampled from $\{0, 1\}^{\lambda}$. Let $E_K(m)$ denote encrypting m with key k and $D_k(c)$ denote decrypting c with key k. Let $H(k, m)$ denote a keyed hash function that takes as input message m and key k and outputs a λ bits string. Let $\mathbf{W}$ denote the keyword set stored on the client side. $|\mathbf{W}|$ denotes the number of distinct keywords stored on the client side.

B. MULTIPLE CLOUD MODEL

With the development of cloud technology, we can divide our database into small-scale databases stored in different clouds that are non-colluding. In this paper, we use three clouds, i.e, cloud c_1, c_2 and c_3. Cloud c_1 and c_2 are used to store documents and the indexes of keywords, while cloud c_3 is used for temporary storage to store the searched documents.

When the client performs a keyword search, we will search from cloud c_1 and c_2 and then store the searched documents in cloud c_3. That is, we do not write these searched documents back immediately. If we do so, the malicious servers (c_1 and c_2) can observe the connection between the searched trapdoor and documents. On the contrary, we store the searched documents in cloud c_3. At some point in the future, we can then write the part of the stored documents back to the cloud c_1 and c_2. Therefore, c_1 and c_2 only get a part of the search pattern leakage. Furthermore, with c_3, we can break the linkability between the searched keyword w and the documents which match w.

C. SEARCHABLE SYMMETRIC ENCRYPTION

In order to use cloud resources, the client can store encrypted documents in the cloud; this brings about the challenge of how to search the encrypted document's matching keyword w. The SSE scheme can solve this problem. When a client wants to search a keyword w, it computes a search token t matching w and then sends t to the server. The server can compute and retrieve the document's identifier with token t and then send these identifiers to the client side. According to these identifiers, the client can download those documents matching w.

A dynamic searchable encryption scheme always consists of one algorithm and two protocols between clients and server:

- Setup(DB): This is an initialization algorithm that takes database DB as an input and gets (EDB, K, σ) as an output, in which EDB is the encrypted database, K is a secret key stored by the client and σ is the state of the client.
- Search(K, q, σ; EDB): On the client side, the protocol takes the key K and its state σ as the input and outputs a query q about the keyword w. On the server side, the protocol takes EDB as the input and outputs the document identifiers matching keyword w. In this paper, we only consider searching a single keyword operation.
- Update(K, σ, op, in; EDB): Adding and deleting a document matching keyword w belong to the update operations. When the client adds a document, the op will be set to add; otherwise, it will be set to del. The client takes the key K, operation op, state σ and in as input. Generally speaking, the client will generate a new block and upload it to EDB on the server side.

D. FORWARD PRIVACY SECURITY

We borrow the formal definition of forward-secure in [2]. If the update operation does not leak any other information more than itself, the scheme is forward private. In particular, the server cannot determine whether the updated document matches a keyword we queried before.

Definition 2: An $\mathcal{L}$-adaptive-secure SSE scheme is forward private if the leakage during the update operation can be written as

$$\mathcal{L}^{Updt}(op, in) = L'(op, (ind_i, u_i)),$$

where (ind_i, u_i) is the modified documents with the matching keyword.

E. BACKWARD PRIVACY SECURITY

Backward privacy makes sure that the server can learn less on keyword w. Generally speaking, for a keyword/document pair (w, ind), it is added and then later deleted from the database. When searching keyword w, the result does not reveal ind [39]. TimeDB(w) denotes a timestamp list matching keyword w. When inserting a document into the database, we record a timestamp in TimeDB(w). When deleting documents, we delete the timestamp of when they were inserted. Formally speaking, TimeDB(w) can be defined as:

$$\text{TimeDB} = \{(u, ind)|(u, add, (w, ind)) \in Q \text{ and } \forall u', (u', del, (w, ind)) \notin Q\},$$

where Q denotes the query list. Then, $DB = \{ind|\exists s.t.(u, ind) \in TimeDB(w)\}$.

Updates(w)is a list of timestamps of updates on w. It can be defined as:

$$Updates(w) = \{u|(u, add, (w, ind)) \text{ or } (u, del, (w, ind)) \in Q\},$$

where Q denotes the query list.

DelHist(w) denotes the list of timestamps for the whole deletion matching keyword w. It can be defined as:

$$\text{DelHist(w)} = \{(u^{add}, u^{del})|\exists \text{ ind s.t. } (u^{del}, add, (w, ind)) \in Q\}.$$

With these notions introduced here, we can describe three levels of backward private.

Definition 3: For a $\mathcal{L} - adaptively - secure$ SSE scheme, if the leakage function of search $\mathcal{L}^{Srch}$ and update $\mathcal{L}^{Updt}$ can be written as follows, this scheme is an insertion pattern revealing backward-private.

$$\mathcal{L}^{Srch}(w) = \mathcal{L}''(TimeDB(w), a_w),$$
$$\mathcal{L}^{Updt}(op, w, ind) = \mathcal{L}'(op),$$

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

For a $\mathcal{L} - adaptively - secure$ SSE scheme, if the leakage function of search $\mathcal{L}^{Srch}$ and update $\mathcal{L}^{Updt}$ can be written as follows, this scheme is an update pattern revealing backward-private.

$$\mathcal{L}^{Srch}(w) = \mathcal{L}''(TimeDB(w), Updates(w)),$$
$$\mathcal{L}^{Updt}(op, w, ind) = \mathcal{L}'(op, w),$$

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

For a $\mathcal{L} - adaptively - secure$ SSE scheme, if the leakage function of search $\mathcal{L}^{Srch}$ and update $\mathcal{L}^{Updt}$ can be written as follows, this scheme is weakly backward-private.

$$\mathcal{L}^{Srch}(w) = \mathcal{L}''(TimeDB(w), DelHist(w)),$$
$$\mathcal{L}^{Updt}(op, w, ind) = \mathcal{L}'(op, w),$$

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

IV. OUR CONSTRUCTION

In this section, we give a detailed description of our construction named "M-SSE", a forward and backward secure searchable encryption scheme based on the multi-cloud technique. It not only supports add-and-delete operations but also can defend against adaptive file injection attacks.

A. STORAGE STRUCTURE

Our M-SSE adopts inverted index schemes like $\Sigma o\varphi o\varsigma$. Let L_w denote the indexed list storing the identifiers $(ind_0, ind_1, \ldots, ind_{n_w})$ of documents which contain keyword w. Furthermore, the size of L_w is n_w.

We adopt three clouds, i.e., cloud c_1, c_2 and c_3. One cloud (c_3) is used as temporary storage, but the others (c_1 and c_2) are used as the normal servers supporting the keyword search. For the cloud c_1 and c_2, the client uploads encrypted documents and keyword ciphertexts to them. But for a document, the client will randomly select a unique cloud to store it. When performing search operations, the client sends two different tokens to both the cloud c_1 and c_2 at the same time. For cloud c_3, it never supports a keyword search because it is only designed to be temporary storage. After keyword w is searched, we obtain the inverted list L_w and all the documents containing keyword w. For security considerations, we will not update these data to cloud c_1 and c_2 immediately; on the contrary, we will encrypt these documents and then cache them to cloud c_3.

On the client side, we adopt a map W to store the state of each keyword. The state of keyword w can be denoted as the tuple $st_w = (token_l, token_r, tag)$, where $token_l$ is the token of the inverted list on cloud c_1 and $token_r$ is that on cloud c_2. *Tag* is a label to show whether documents matching keyword w are on c_3. For each keyword $w \in W$, the map W stores its state st_w.

Figure 2 shows the structure of M-SSE. Unlike $\Sigma o\varphi o\varsigma$, we store $E_{k_w}(ind, op)$ instead of (ind, op), where op can be

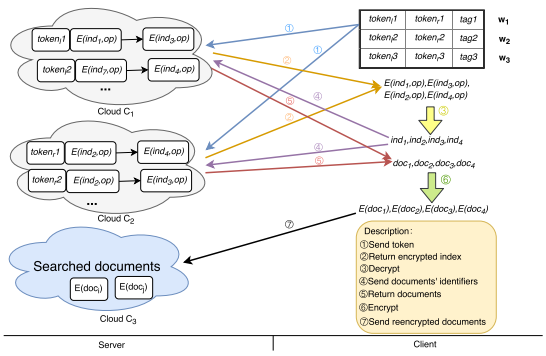


FIGURE 2. Framework of M-SSE. There are three clouds in the scheme: cloud c_1 and c_2 store documents and indexes as other SSE paradigm; c_3 is a temporary storage that stores the searched documents.

add or *del*, meaning add or delete operations, respectively. This approach provides backward privacy.

B. OUR CONSTRUCTION

Algorithm 1 shows the formal description of our scheme, M-SSE. The scheme supports both add and delete operations, and the client sends these data blocks with different operations to cloud c_1 or c_2. In our scheme, **H** is a keyed hash function whose output is μ bits long; furthermore, $E_k(m)$ and $D_k(c)$ are implemented by an IND-CPA symmetric encryption.

Algorithm 1 M-SSE Basic Construction of Multi-Cloud SSE

Setup()

$K_\Sigma \xleftarrow{\$} \{0,1\}^\lambda$
$(SK, PK) \xleftarrow{\$} KeyGen(1^\lambda)$
$W, T \leftarrow$ *empty map*

Search(w, σ; EDB)

Client:

$K_w \leftarrow F(K_\Sigma, w)$
$(ST_c, c) \leftarrow W[w]$
If $(ST_c, c) = \perp$ *and Tag* $\neq \perp$
  documents matching w are stored in c_3
Else
  Send $(K_w, ST_c.token_l, c_l)$ *to* c_1, $(K_w, ST_c.token_r, c_r)$ *to* c_2
Cloud c_1 *and* c_2:
Run $\Sigma o\varphi o\varsigma - B.Search(w, \sigma; EDB)$ *and get* $E_{K_S}(ind, op)$,
store the result in S
Send S to the client.
Client:
Decrypt S and get $\{ind : \exists\, i, (ind_i, op_i) \cap \forall j > i, (ind_j, op_j) \neq (ind, del)\}$

Update(w,c_1, c_2; EDB)

Client or cloud c_3:

$K_S \xleftarrow{\$} \{0,1\}^\lambda$
Randomly select documents sending to c_1 and c_2 respectively.
Run $\Sigma o\varphi o\varsigma - B.Update(w, E_{K_S}(ind, op), \sigma; EDB)$ on cloud c_1 and c_2

1) UPDATE OPERATION

When updating a document matching keyword w, the client will generate a new data block and randomly send it to cloud c_1 or c_2. Adding and deleting a document are implemented with the update operation. As for the add operation, the document can be a fully new document and also a document stored in c_3, which has been searched and downloaded. As for the delete operation, the document must have already been stored in the database. The following is a detailed approach:

Step 1 (Select Target Cloud Randomly): Select the cloud that the new block will be sent to. The new block contains the encryption of the new document identifier and the operation. Therefore, a new block has two possible locations and thus it support more privacy.

Step 2 (Generate a New Token): According to the result of *step 1*, the client generates a new token for the new block. The token has a corresponding relationship with the new block matching keyword w. So the one who obtains the token can get the corresponding document matching w.

Step 3 (Update the Map W): If the new block is sent to cloud c_1, $W[w].ST_w.token_l$ and $W[w].c_l$ will be updated with the new value; otherwise, $W[w].ST_w.token_r$ and $W[w].c_r$ will be updated with the new value.

2) SEARCH OPERATION

The client generates a search token t corresponding to keyword w. Search token t will allow the server to retrieve document identifiers matching keyword w. Following is the detailed approach:

Step 1 (Retrieve Map W): According to keyword w, we will get the state of w. If $W[w].ST_c.token_l$ is not $\perp$, we will send $W[w].ST_c.token_l$ to cloud c_1. Otherwise, we will send $W[w].ST_c.token_r$ to cloud c_2. If $W[w].ST_c.token_l$ and $W[w].ST_c.token_r$ are $\perp$, we will check whether the documents match w in cloud c_3.

Step 2 (Merge the Identifier): The data blocks received from clouds are encrypted with K_S, so the client will decrypt these blocks first. For any ind_i, if there exist both (ind_i, add) and (ind_i, del), we will ignore this ind_i.

Step 3 (Get the Documents): For any $ind_i = D_{K_w}(E_{K_w}(ind_i, op))$, according to the position map, we can know the position of the documents. Then, sending the corresponding *ind* from step 2 to different clouds can get the documents.

Step 4 (Upload the Searched Result to Cloud c_3): Once we have searched keyword w and got the documents, we will then send the encrypted documents and identifiers to c_3 and reset *Tag*. After a while, we will update these documents to c_1 and c_2 as new documents.

C. SECURITY ANALYSIS

We can use the Random Oracle Model to prove the security of M-SSE.

Theorem 1 (Adaptive security of M-SSE): Define $\mathcal{L}_S = (\mathcal{L}_S^{Search}, \mathcal{L}_S^{Update})$, where

$$\mathcal{L}_S^{Search} = (\mathtt{sp}(w), Hist(w)), \quad \mathcal{L}_S^{Update}(op, w, ind) = \perp .$$

M-SSE is $\mathcal{L}_S - adaptive - secure$.

Proof: Deriving several games from $SSEReal_A^{M-SSE}(\lambda)$, which is a real-world game that can help prove the theorem.

Game G_0: G_0 is the real world SSE security game *SSEReal*. Formally speaking,

$$P[SSEReal_A^{M-SSE}(\lambda) = 1] = P[G_0 = 1].$$

Game G_1: When confronting a new keyword w, G_1 picks a new key randomly instead of calling F when generating

K_W. Furthermore, K_W will be stored in a table key, so the next time the same keyword w is confronted, we get the K_W from key. Therefore, if there is an adversary that can build a reduction that is able to distinguish between F and a truly random function, we can say that the adversary can distinguish between G_0 and G_1. Formally speaking,

$$P[G_0 = 1] - P[G_1 = 1] \leq Adv_{F,B}^{PRF}(\lambda),$$

where B is an efficient adversary.

According to the proof in $\Sigma o\varphi o\varsigma$, we can conclude that M-SSE is $\mathcal{L}_S - adaptive - secure$.

Theorem 2 (M-SSE can Protect Size Pattern): Define $c.num = (c_1.num, c_2.num)$, where $c_1.num$ is the number of documents matching keyword w in cloud c_1 and $c_2.num$ is the number of documents matching keyword w in cloud c_2. M-SSE can protect the size pattern.

Proof: We divide an encrypted database into two small databases and store them in cloud c_1 and c_2. When performing a search operation, we send two tokens to c_1 and c_2. Then, we merge the index and request document matching w. For c_1, it only knows the number $c_1.num$ of document matching w stored in c_1; for c_2, it only knows the number $c_2.num$ of document matching w stored in c_2. The total number c.num of documents matching w is $c_1.num+c_2.num$. For non-collusive clouds, they only know the number of documents stored on them except the total number. Formally speaking,

$$P[c_2 \ knows \ c_1.num] = negl(\lambda),$$
$$P[c_1 \ knows \ c_2.num] = negl(\lambda),$$

where $negl(\lambda)$ is the negligible function.

Therefore, we can say that with the support of the multi-cloud technique, M-SSE can protect the size pattern.

D. LINKABILITY ANALYSIS

When searching keyword w, the client sends a trapdoor $Token_l$ and $Token_r$ to the server and gets the index from the clouds; then, the client can get the documents D matching w from c_1 and c_2. If the client updates these documents to c_1 and c_2 immediately, the server can know w, $Token_l$, $Token_r$, and the documents D are related. In order to reduce the linkability between w and the searched documents that match w, we use another cloud c_3 as a temporary storage to store the documents that match w and have already been searched. In this way, when searching keyword w', the client sends $Token_l'$ and $Token_r'$ to the server and then gets the index. The client encrypts the index and gets the true identifiers of the documents and then asks the server for these documents D'. Next, these documents are sent to c_3 temporarily. Therefore, the server cannot know that w', $Token_l'$, $Token_r'$ and documents D' are related.

V. EXPERIMENTS AND EVALUATIONS

Our experiment focuses on the comparison of efficiency between M-SSE and other typical SSE schemes. In the M-SSE scheme, the search operation can be divided into two parts; one is generating the token on the client side, and the other is searching on the server side. The update operation can also be divided into two parts; one is generating a new block on the client side, and the other is uploading this new block to the server side. We divide these operations into two parts and observe the bandwidth, computation on the server side and client side.

A. IMPLEMENTATION DETAILS

We implement the core function and benchmark of M-SSE with C/C++. The cryptographic primitives in M-SSE use the code provided by $\Sigma o\varphi o\varsigma$ [2]'s source code. We use HMAC as the keyed hash function and use the OpenSSL's BigNum library to implement RSA. And we use RSA to implement trapdoor permutation.

1) EXPERIMENT ENVIRONMENT

For server storage, we use RocksDB to store the map. Our experiment is run on a desktop computer; it has single Inter Core i7-7700 3.60HZ CPU, 2GB of RAM on ubuntu 14.0.4.

2) PARAMETER

We set the secure parameter λ to 128 bits. The maximum number of keyword/document pairs range from 140 to 14000000, which is determined by concrete benchmarks. For symmetric primitives, cryptographic keys are 128 bits long, and the length of RSA keys is 2048 bits long.

B. EVALUATION

We evaluated the performance of M-SSE with 140000 keyword-document pairs. Three operations are considered in the experiment: token generation, search operation and update operation.

1) TOKEN GENERATION

As shown in Figure 3, during the search operation, we evaluate the performance of token generation. To the best of our knowledge, Fides has the best performance in terms of token generation. M-SSE and Fides are all based on the $\Sigma o\varphi o\varsigma$ scheme. While generating tokens, M-SSE and Fides perform the same operation, so the performance of token generation

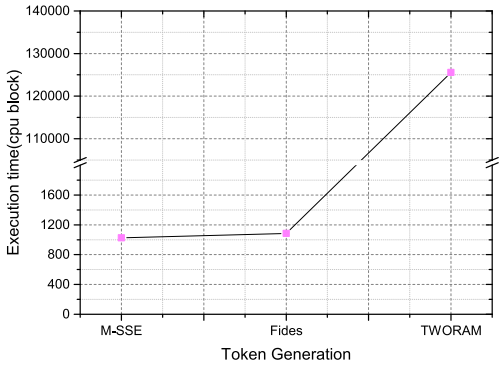


FIGURE 3. Comparison of token generation.

is almost the same. The result of the experiment shows that the speed of M-SSE is not worse than that of Fides, and it is almost the same as Fides. However, M-SSE protects more privacy than Fides. Figure 3 also shows that TWORAM is the most inefficient one although it leaks the minimum information.

2) SSE OPERATION

As shown in Figure 4 and Figure 5, we compare the performance of search and update operations with other schemes. For M-SSE and Fides, RSA operations will not be fully interleaved with disk accesses at the beginning of the search operation. However, mutexes and storage accesses will induce latency. M-SSE stores the index matching keyword w on two clouds; thus we can get the index simultaneously. Therefore, the speed of the search operation is nearly $2\times$ better than that of Fides. For TWORAM, because of the complexity of ORAM itself, the efficiency of the search operation and update operation are much lower. We test these schemes with the same keyword/document pairs, keyword sets and benchmarks. Therefore, the initialization and benchmarks are the same. We can conclude that M-SSE has the same performance as Fides and is much better than TWORAM.

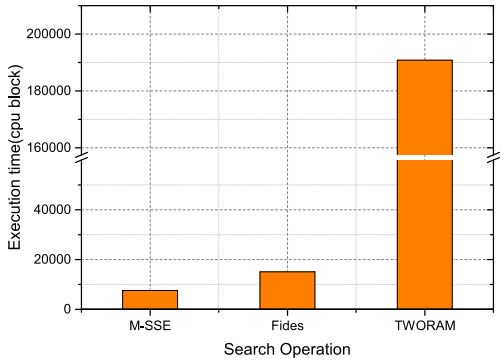


FIGURE 4. Comparison of Search operations.

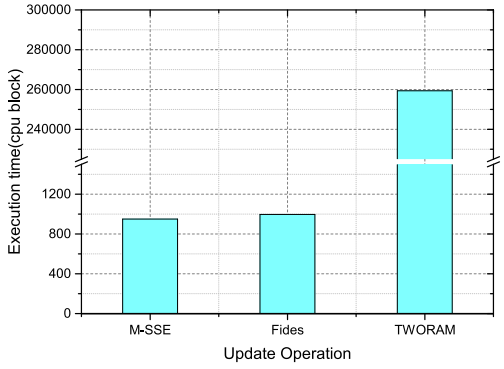


FIGURE 5. Comparison of Update operations. Keyword-document pairs is set to 140000.

3) SECURITY LEVEL

With the support of the multi-cloud technique, M-SSE can distribute the leakages to different clouds. If the clouds are non-collusive, M-SSE can reduce the information known by each cloud. Therefore, M-SSE can protect the size pattern. Furthermore, M-SSE can support forward-privacy, so it can defend against file injection attacks. Based on the backward-private level, M-SSE realizes backward privacy with the update pattern.

M-SSE has nearly the same optimal performance as Fides, which has the best performance of SSE, but it provides a higher secure level. TWORAM leaks the minimum privacy, but the performance is much worse than that of M-SSE. Reducing the leakages from secure searchable encryption is the main point of M-SSE. Therefore, we can say that M-SSE strikes a good balance between efficiency and security.

VI. CONCLUSION

In this paper, we focus on how to improve the performance of the SSE scheme and reduce its leakages. Based on non-colluding clouds, we propose the M-SSE scheme, which achieves both forward and backward security. Apart from the good performance, M-SSE can protect the size pattern. Distributing the leakages to different clouds to reduce the information leakage may be a new idea to protect users' privacy. In future work, we plan to design more secure searchable encryption with better performance and the forward secure order-preserving encryption scheme.

REFERENCES

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

[2] R. Bost, "$\Sigma o\varphi o\varsigma$: Forward secure searchable encryption," in *Proc. ACM CCS*, 2016, pp. 1143–1154.

[3] R. Bost, P.-A. Fouque, and D. Pointcheval, "Verifiable dynamic symmetric searchable encryption: optimality and forward security," Int. Assoc. Cryptol. Res., Las Vegas, NV, USA, Tech. Rep. 2016/062, 2016, vol. 62. [Online]. Available: https://eprint.iacr.org/2016/062

[4] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM CCS*, 2017, pp. 1465–1482.

[5] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. CCS*, 2015, pp. 668–679.

[6] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *Proc. CRYPTO*, 2013, pp. 353–373.

[7] D. Cash *et al.*, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proc. NDSS*, vol. 14. 2014, pp. 23–26.

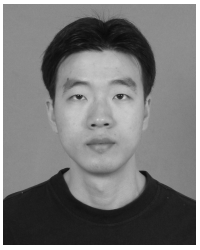
[8] X. Chen, J. Li, J. Weng, J. Ma, and W. Lou, "Verifiable computation over large database with incremental updates," *IEEE Trans. Comput.*, vol. 65, no. 10, pp. 3184–3195, Oct. 2016.

[9] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. S. Wong, "New algorithms for secure outsourcing of large-scale systems of linear equations," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 1, pp. 69–78, Jan. 2015.

[10] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, Jan. 2011.

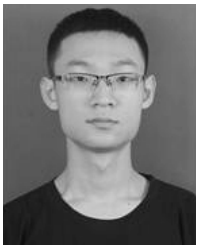
[11] C. Gao, Q. Cheng, P. He, W. Susilo, and J. Li, "Privacy-preserving Naive Bayes classifiers secure against the substitution-then-comparison attack," *Inf. Sci.*, vol. 444, pp. 72–88, May 2018.

[12] C. Gao, Q. Cheng, X. Li, and S.-B. Xia, "Cloud-assisted privacy-preserving profile-matching scheme under multiple keys in mobile social network," *Cluster Comput.*, pp. 1–9, Feb. 2018, doi: 10.1007/s10586-017-1649-y.
[13] S. Garg, P. Mohassel, and C. Papamanthou, "TWORAM: Round-optimal oblivious RAM with applications to searchable encryption," in *Proc. Annu. Cryptol. Conf.*, 2016, pp. 563–592.
[14] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," in *Proc. JACM*, 1996, pp. 431–473.
[15] M. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. NDSS*, vol. 20, 2012, p. 12.
[16] R. H. Jhaveri, N. M. Patel, Y. Zhong, and A. K. Sangaiah, "Sensitivity analysis of an attack-pattern discovery based trusted routing scheme for mobile ad-hoc networks in industrial IoT," in *IEEE ACCESS*, vol. 6, pp. 20085–20103, 2018, doi: 10.1109/ACCESS.2018.2822945.
[17] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Proc. EUROCRYPT*, 2017, pp. 94–124.
[18] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. CCS*, 2016, pp. 965–976.
[19] K. Kurosawa and Y. Ohtaki, "UC-secure searchable symmetric encryption," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, vol. 7397, 2012, pp. 285–298.
[20] J. Li, X. Chen, S. S. M. Chow, Q. Huang, D. S. Wong, and Z. Liu, "Multi-authority fine-grained access control with accountability and its application in cloud," *J. Netw. Comput. Appl.*, vol. 112, pp. 89–96, Jun. 2018.
[21] J. Li *et al.*, "Secure distributed deduplication systems with improved reliability," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3569–3579, Dec. 2015.
[22] J. Li, X. Chen, M. Li, J. Li, P. P. C. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1615–1625, Jun. 2014.
[23] J. Li, X. Chen, F. Xhafa, and L. Barolli, "Secure deduplication storage systems supporting keyword search," *J. Comput. Syst. Sci.*, vol. 81, no. 8, pp. 1532–1541, 2015.
[24] B. Li, Y. Huang, Z. Liu, J. Li, Z. Tian, and S.-M. Yiu, "HybridORAM: Practical oblivious cloud storage with constant bandwidth," *Inf. Sci.*, pp. 1–13, Feb. 2018, doi: 10.1016/j.ins.2018.02.019.
[25] T. Li, J. Li, Z. Liu, P. Li, and C. Jia, "Differentially private Naive Bayes learning over multiple data sources," *Inf. Sci.*, vol. 444, pp. 89–104, May 2018.
[26] J. Li, Z. Liu, X. Chen, X. Tan, and D. S. Wong, "L-EncDB: A lightweight framework for privacy-preserving data queries in cloud computing," *Knowl.-Based Syst.*, vol. 79, pp. 18–26, May 2015.
[27] H. Li, D. Liu, Y. Dai, T. H. Luan, and X. S. Shen, "Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 1, pp. 127–138, Mar. 2015.
[28] J. Li, L. Wang, L. Wang, Z. Huang, and J. Li, "Verifiable Chebyshev maps-based chaotic encryption schemes with outsourcing computations in the cloud/fog scenarios," *Concurrency Comput., Pract. Exper.*, pp. 1–10, Jun. 2018, doi: 10.1002/cpe.4523.
[29] H. Li, Y. Yang, Y. Dai, S. Yu, and Y. Xiang, "Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data," *IEEE Trans. Cloud Comput.*, pp. 1–11, Nov. 2017, doi: 10.1109/TCC.2017.2769645.
[30] Q. Lin, J. Li, Z. Huang, W. Chen, and J. Shen, "A short linearly homomorphic proxy signature scheme," *IEEE Access*, vol. 6, pp. 12966–12972, 2018.
[31] Q. Lin, H. Yan, Z. Huang, W. Chen, J. Shen, and Y. Tang, "An ID-based linearly homomorphic signature scheme and its application in blockchain," *IEEE Access*, vol. 6, pp. 20632–20640, 2018.
[32] Q. Liu, Y. Guo, J. Wu, and G. Wang, "Effective query grouping strategy in clouds," *J. Comput. Sci. Technol.*, vol. 32, no. 6, pp. 1231–1249, Nov. 2017.
[33] Z. Liu, Y. Huang, J. Li, X. Cheng, and C. Shen, "DivORAM: Towards a practical oblivious RAM with variable block size," *Inf. Sci.*, vol. 447, pp. 1–11, Jun. 2018.
[34] Z. Liu, T. Li, P. Li, C. Jia, and J. Li, "Verifiable searchable encryption with aggregate keys for data sharing system," *Future Generat. Comput. Syst.*, vol. 78, pp. 778–788, 2018.
[35] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Proc. Secur. Privacy*, May 2014, pp. 639–654.
[36] J. Shen, Z. Gui, S. Ji, J. Shen, H. Tan, and Y. Tang, "Cloud-aided lightweight certificateless authentication protocol with anonymity for wireless body area networks," *J. Netw. Comput. Appl.*, vol. 106, pp. 117–123, Mar. 2018.
[37] J. Shen, C. Wang, T. Li, X. Chen, X. Huang, and Z. Zhan, "Secure data uploading scheme for a smart home system," *Inf. Sci.*, vol. 453, pp. 186–197, Jul. 2018, doi: 10.1016/j.ins.2018.04.048.
[38] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. Secur. Privacy*, May 2000, pp. 44–55.
[39] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. NDSS*, vol. 71, 2014, pp. 72–75.
[40] C. Wang, J. Shen, Q. Liu, Y. Ren, and T. Li, "A novel security scheme based on instant encrypted transmission for Internet of Things," *Secur. Commun. Netw.*, vol. 2018, May 2018, Art. no. 3680851, doi: 10.1155/2018/3680851.
[41] J. Xu, L. Wei, Y. Zhang, A. Wang, F. Zhou, and C.-Z. Gao, "Dynamic Fully Homomorphic encryption-based Merkle Tree for lightweight streaming authenticated data structures," *J. Netw. Comput. Appl.*, vol. 107, pp. 113–124, Apr. 2018.
[42] X. Zhang, X. Chen, J. Wang, Z. Zhan, and J. Li, "Verifiable privacy-preserving single-layer perceptron training scheme in cloud computing," *Soft Comput.*, pp. 1–14, May 2018, doi: 10.1007/s00500-018-3233-7.
[43] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: the power of file-injection attacks on searchable encryption," in *Proc. USENIX Secur.*, 2016, pp. 707–720.
[44] X. Zhang, Y.-A. Tan, C. Liang, Y. Li, and J. Li, "A covert channel over VoLTE via adjusting silence periods," *IEEE Access*, vol. 6, pp. 9292–9302, 2018.



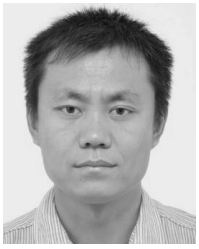
CHONGZHI GAO received the Ph.D. degree in applied mathematics from Sun Yat-sen University in 2004. He is currently a Professor with the School of Computer Science, Guangzhou University. His research interests include cryptography and privacy in machine learning.

SIYI LV received the B.Eng. degree from the College of Computer and Control Engineering, Nankai University, in 2016.



YU WEI is currently pursuing the bachelor's degree in engineering and law with the College of Computer and Control Engineering, Nankai University.

ZHI WANG received the Ph.D. degree in information security from Nankai University in 2012. From 2005 to 2007, he was with Fortinet, Inc., as an Antivirus Engineer. From 2013 to 2015, he was with S2Lab ISG RHUL as a Post-Doctoral Researcher. He is currently with Nankai University as a Lecturer. His research interests include malware analysis and machine learning.



ZHELI LIU received the B.Sc. and M.Sc. degrees in computer science and the Ph.D. degree in computer application from Jilin University, China, in 2002, 2005, and 2009, respectively. After a Post-Doctoral Fellowship at Nankai University, he joined the College of Computer and Control Engineering, Nankai University, in 2011, where he is currently an Associate Professor. His current research interests include applied cryptography and data privacy protection.

XIAOCHUN CHENG (SM'04) received the B.Eng. degree in computer software in 1992 and the Ph.D. degree in artificial intelligence in 1996. He is currently the Secretary for the IEEE SMC UK & RI. He is a member of the IEEE SMC: Technical Committee on Systems Safety and Security. He is also a Committee Member of the European Systems Safety Society.