IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 10, 2019, accepted December 21, 2019, date of publication December 24, 2019,
date of current version January 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2961971

Benchmarking Dynamic Searchable Symmetric Encryption Scheme for Cloud-Internet of Things Applications

YEN-WU TI[1], CHIA-FENG WU[2], CHIA-MU YU[3], AND SY-YEN KUO[2]
[1]College of Artificial Intelligence, Yango University, Hangzhou 310058, China
[2]Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan
[3]Department of Computer Science and Engineering, National Chung Hsing University, Taichung 402, Taiwan

Corresponding author: Chia-Mu Yu (chiamuyu@gmail.com)

ABSTRACT Recently, the rapid development of Internet of things (IoT) has resulted in the generation of a considerable amount of data, which should be stored. Therefore, it is necessary to develop methods that can easily capture, save, and modify these data. The data generated using IoT contain private information; therefore sufficient security features should be incorporated to ensure that potential attackers cannot access the data. Researchers from various fields are attempting to achieve data security. One of the major challenges is that IoT is a paradigm of how each device in the Internet infrastructure is interconnected to a globally dynamic network. When searching in dynamic cloud-stored data, sensitive data can be easily leaked. IoT data storage and retrieval from untrusted cloud servers should be secure. Searchable symmetric encryption (SSE) is a vital technology in the field of cloud storage. SSE allows users to use keywords to search for data in an untrusted cloud server but the keywords and the data content are concealed from the server. However, an SSE database is seldom used by cloud operators because the data stored on the cloud server is often modified. The server cannot update the data without decryption because the data are encrypted by the user. Therefore, dynamic SSE (DSSE) has been developed in recent years to support the aforementioned requirements. Instead of decrypting the data stored by customers, DSSE adds or deletes encrypted data on the server. A number of DSSE systems based on linked list structures or blind storage (a new primitive) have been proposed. From the perspective of functionality, extensibility, and efficiency, these DSSE systems each have their own advantages and drawbacks. The most crucial aspect of a system that is used in the cloud industry is the trade-off between performance and security. Therefore, we compared the efficiency and security of multiple DSSE systems and identified their shortcomings to develop an improved system.

INDEX TERMS Searchable encryption, dynamic searchable encryption.

I. INTRODUCTION

The rapid development of Internet of things (IoT) and cloud computing has resulted in a high demand for cloud storage environments. In these systems, to protect privacy, users query some information multiple times and receive the content of the query, but their identity or the content of the stored message is not revealed. Over the years, several studies have focused on protecting databases from malicious users. An attacker can recreate a valuable message by querying the database. Even if the database has some protective features, the attacker can still succeed in obtaining information. Thus, if a user wants to protect their privacy, the safest method is to download the complete encrypted file stored in the database. The user then decrypts the file and performs the required processing. However, this approach results in a large and unaffordable communication traffic. This is not a viable option. Therefore, we require a method that protects user privacy and does not burden networks and servers. One of the most common solutions to search encrypted data is searchable symmetric encryption (SSE). Searchable encryption (SE) was first introduced in [1], and allows data protection by encryption but can retrieve desired information efficiently using keyword searching. Thus, if we store encrypted documents on cloud storage, the server need not know the data content, and we can still search for files on

The associate editor coordinating the review of this manuscript and approving it for publication was Patrick Hung.

the server and extract them [2]. The server is not expected to know about the keywords and only a trivial message about keywords and files extracted is sent to the server. The conditions appear to be contradictory but it is possible to construct such a system. However, SSE has some drawbacks. In general, the data stored in the database is dynamic, that is, it changes constantly because users often edit the files stored in the server through the network. Therefore, it is necessary to increase the functionality of SSE to enable users to update and delete the contents of encrypted documents. This is known as dynamic searchable symmetric encryption (DSSE).

In the past, a number of different DSSE schemes have been developed, which, in addition to meeting the basic needs of adding and deleting data, have unique characteristics. The DSSE scheme [3] can complete the search in sublinear time and is resistant to adaptive chosen-keyword attacks. The first DSSE that was used on large databases was proposed by Cash *et al.* in 2014 [4]. To the best of our knowledge, the DSSE scheme proposed by Stefanov *et al.* in 2014 provides the highest security [5]. In addition, some schemes cater to special requirements in practice. For example, the DSSE scheme in [6] limits the cloud server to be a pure storage system, without allowing the client to perform computing tasks on the cloud serve; some schemes can operate without the client storage capacity [7].

In order to prevent malicious attackers from reading untrusted memory and getting data to cause users to leak sensitive information while storing and reading data, oblivious random access machines (ORAM) scheme is a common solution. The ORAM approach is to separate the operation of the memory access from other system operations, so that the malicious attacker does not know which file the CPU is using. It is even hard to know that the CPU is using the same file repeatedly. So its most important application is to avoid access patterns being detected by third parties. The new ORAM solution has been able to keep the client's storage down to a constant level. However, since most ORAM schemes use Homomorphic Encryption, their traffic may reach the logarithmic scale, and multiple repeated communications are required, which still leads to expensive overhead. Especially for large databases, the high cost of ORAM and Homomorphic Encryption is hard to ignore [3]–[7].

A. RESEARCH MOTIVATION

Most schemes claim to be efficient, but no experimental comparison has been conducted on the performances of various schemes. Therefore, in this study, a number of representative DSSE schemes were implemented and their performances were compared. The same standard was used to investigate the advantages and disadvantages of various DSSE schemes in a DSSE system. The notation of security definitions and present SSE concepts are presented in Section 2.

B. RESEARCH INTENTION

We discuss the trade-off between efficiency and security for a specific SE scheme that allows a single client to store encrypted data with a private key on a remote server, and the client can send a single keyword query to the server. The server returns the search result to the client without decrypting the data. The client can also add or delete file queries to the server, causing the server to add the new file or delete an existing file while only learning trivial information about the files (i.e., we focus on DSSE systems).

This paper was published in part in Proc. International Conference on Intelligent Computing and Its Emerging Applications (ICEA) [2], 2019.

II. BACKGROUND

This section provides background introduces DSSE, index, privacy issues, and security definitions.

A. DYNAMIC SEARCHABLE SYMMETRIC ENCRYPTION

The IoT industry and cloud service providers rely on remote computing and storage resources to serve customers. These services are often used to backup data or reduce local operational costs. However, the remote server cannot ensure absolute data security. The data can be accessed by the server administrator or hackers with root permissions. Therefore, if sensitive data are to be stored in an untrusted server, the data should be encrypted to ensure attackers and administrators cannot access plaintext data without appropriate keys. Although this method increases security and reduces privacy risks, a simple search is not possible. Before 2000, only two solutions were available to search encrypted data: either the user downloaded the whole database from the server, then decrypted and searched or allowed the server to decrypt the data, search the plaintext data, and return the result to the user. The first method is impractical and requires high overheads (local storage space, communication, and computation); the second method renders the encryption meaningless because the server can access the information from the plaintext data. However, in 2000, Song *et al.* [1] proposed the SE technique for searching encrypted data, in which the server can support the search function without decrypting the data, with a small information loss.

Although SE schemes were originally based on the client-server model, we can still classify them under user scenarios or key primitives. For user scenarios, we classify them by the number of clients. If we only have one client in an SE scheme, then only a single user can write encrypted data to the server and send a search query to read the data from the server. This is the single writer/single reader (S/S) scenario. Similarly, we have three other user scenarios: multi-writer/single reader (M/S), single writer/multi-reader (S/M), and multi-writer/multi-reader (M/M). Two key primitives are available, namely symmetric and asymmetric. Symmetric key primitives use a private key to encrypt and decrypt data, which can be used to devise S/S SE schemes. Asymmetric key primitives use a public key to encrypt, and a private key to decrypt, which allows M/S SE schemes.

However, SE schemes are not sufficiently practical for real use. Generally, the data are dynamic, that is, edited often even

when on the remote server. To satisfy this usage scenario, SE was extended to dynamic SE (DSE), which provided the server ability to add, update, and delete encrypted data without decryption. A DSE scheme with symmetric key primitives is called DSSE.

SE research involves three parameters, namely efficiency, security, and query expression. We can measure efficiency using complex computations and communications, implement an SE scheme and test run time directly. For DSE, the efficiency of adding or deleting files should be measured. Security of an SE scheme can be defined formally and discussed under various security models. The security definition must be modified for DSE. The query expression for an SE scheme defines what search queries can be used, for example, a single keyword search or conjunctive keyword search, among others. A DSE scheme cannot simultaneously achieve high efficiency, security, and complex query expression; some trade-off should be considered. We focus on the trade-off between DSSE security and efficiency in this paper.

1) GENERAL MODEL FOR A DSSE SCHEME

General Model for a DSSE Scheme: A DSSE scheme allows a user to store encrypted data on a remote server. The server can search the encrypted data without learning information about the plaintext data. The user can also add or delete files on the remote server, with the server only knowing the amount of information, rather than any details. Generally, this capability is achieved by the user by generating an encrypted search index, and storing this index with the encrypted data on the remote server.

Figure 1 shows the general model for an index based DSSE scheme. Suppose we a file set $F = (f_1, f_2, \ldots, f_n)$ with n files and a corresponding index $W = (w_1, w_2, \ldots, w_m)$ with m distinct keywords, where W is extracted from $f_1, f_2, \ldots, f_n$. This file set is to be stored to a remote server using a DSSE scheme. First, the client generates a searchable encrypted index, I, of F using the BuildIndex algorithm with key K. Then, the client uses the Enc algorithm to encrypt each file in F with key K' and outputs a ciphertext, C. Finally, the client uploads I and C to an untrusted (honest-but-curious [8]) server that complies with our protocol, but attempts to learn information about the encrypted data. The encrypted data stored on the server has the form

$$I = \text{BuildIndex}_K(F, W) \quad \text{and } C = Enc_{K'}(f_1, f_2, \ldots, f_n)$$

To search a keyword $w \in W$, the client generates token T using the SrchToken algorithm, and sends T to the server. The server uses T as the input to the Search algorithm, searches I to find files that include w, and returns the appropriate (encrypted) file set, F_w, to the client, where each file $f_i \in F_w$ includes w. The client receives the encrypted file set F_w, and uses the Dec algorithm with key K' to recover the plaintext.

In DSSE schemes, the client can also add or delete files on the remote server. If the client wants to add a new file, f_{new} such as $\notin F, f_{new} \subseteq W$, then the client generates an add token, T_a, using the AddToken algorithm, and upload T_a and $C_a = Enc_{K'}(f_{new})$ to the server. The server adds the new file using the Add algorithm, and the Search algorithm covers the new file.

Deleting a file is a similar operation. The client generates a delete token, T_d, using the DelToken algorithm for deleting an existing file $f_{old} \in F$, and the server deletes the file using the Del algorithm.

B. EFFICIENCY AND INDEX

To reduce the search response time, that is, improve efficiency, the useful and common technique index in database system is used. Each DSSE scheme discussed in this paper (with the exception of [1]) uses index to speed up the search or build process. Forward and invert indexes can be obtained from the plaintext data, as illustrated in Figure 2 for unencrypted data. We use this simple example to illustrate the various DSSE schemes discussed here.

1) FORWARD INDEX

A forward index records the keywords contained within a file, that is, one index per file. Forward index can reduce search times to the number of files. If we have n files, then search complexity is O(n) because the server searches each index for a given keyword.

2) INVERT INDEX

In contrast to the forward index, an invert index records file IDs that include the given keyword. Thus, if we have m keywords, search complexity is O(m). Hash trees can be used here, which provides further speed up.

C. PRIVACY PROBLEM (LEAKAGE)

DSSE schemes leak information (leakages) during search or update (add, delete) operations. The following five privacy issues occur:

1) INDEX INFORMATION

Index Information: Index information is leaked from the encrypted index or ciphertext. This includes the number of keywords or files, file sizes, file IDs, among others.

2) SEARCH PATTERN

Search pattern information allows the server to derive whether two search queries use the same keyword. If the DSSE scheme uses a deterministic algorithm to generate a search token, then the search pattern is also leaked, because for the same keyword the user generates the same search token. However, we proposed a simple method to hide the search pattern in section 4.

3) ACCESS PATTERN

Access pattern information can be obtained from the search result by the server (e.g. file ID or memory access location).

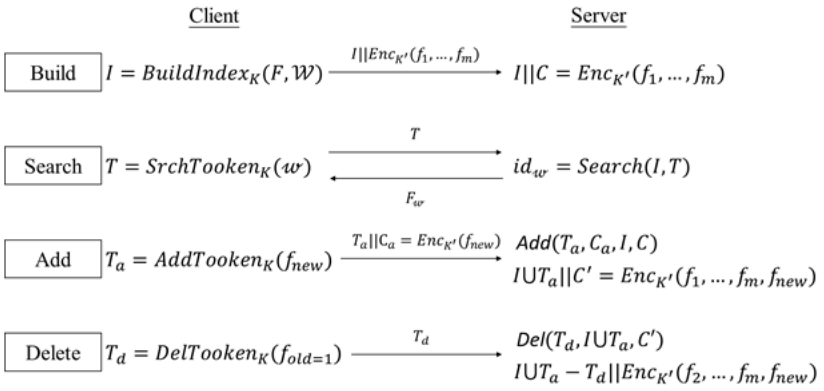


FIGURE 1. General model for an index-based DSSE scheme.

File id	Keywords
1	w_1
2	w_1, w_2, w_3
3	w_2, w_3

(a) Forward index.

Keywords	File id
w_1	1, 2, 3
w_2	2
w_3	2, 3

(b) Invert index.

FIGURE 2. A small example of an unencrypted forward and invert index.

4) FORWARD PRIVACY

Forward privacy denotes that if we searched a keyword, w, previously, then the server cannot know that whether a new file, added subsequent to the search, contains w.

5) BACKWARD PRIVACY

Backward privacy denotes that we cannot execute queries over deleted files. No current DSSE scheme achieves backward privacy.

D. SECURITY DEFINITION

We review SE development and security definition, and provide simplified explanations.

1) IND-CPA

In 2000, Song *et al.* proposed the first SE scheme, SWP [1], formal security definitions were not introduced for the SE, although the authors showed that their scheme was a secure pseudo-random generator. Subsequently, Kamara *et al.* [3] revealed that SWP was indistinguishable against chosen plaintext attacks (IND-CPA). If an encryption scheme was IND-CPA, an adaptively adversary, A cannot distinguish two arbitrary ciphertexts selected by itself (i.e., can query the encryption oracle). That is, the ciphertexts do not leak information about the plaintext. The main leakage does not occur from the ciphertext, but rather the token, and IND-CPA security does not address this concern. Therefore, IND-CPA security is not an appropriate security definition for SE schemes.

2) IND1/2-CKA

The first security notation for SE was introduced by Goh [9], who defined semantic security for secure indexes. This is indistinguishable against adaptive chosen-keyword attacks (IND1-CKA). IND1-CKA guarantees that A cannot learn plaintext information from the index and ciphertext, with the exception of the number of files and their length. Thus, if two encrypted file sets exist with the same size and an encrypted index, adversary A cannot know which index was built from which file set. Chang and Mitzenmacher [10] proposed a superior simulation-based IND-CKA security definition to protect the file size. Goh proposed IND2-CKA to protect the file size. These studies guarantee that A cannot differentiate indexes from two encrypted file sets, even if they have different sizes. However, neither IND1-CKA nor IND2-CKA security definitions provide security for the token. IND1-CKA nor IND2-CKA do not guarantee that the server is unable to recover the plaintext from the token, and so they are considered weak for SE schemes.

3) IND-CKA1/2

Because the previous security definitions were inadequate for SSE schemes, and index security is associated with token security, Curtmola *et al.* [11] proposed two new adversarial models for SSE schemes: nonadaptive (IND-CKA1) and adaptive (IND-CKA2) security. The two definitions define that nothing should be leaked from ciphertext, and both protect token security. Thus, A cannot learn information about keywords from a search token unless it learns from the search or access pattern. However, IND-CKA1 only guarantees security if all queries are generated at once by the user. Thus, IND-CKA2 has been widely used and is considered a strong security definition for SSE schemes.

4) OTHER MODELS

Other security definitions for SSE scheme in different models include universal composability (UC) [12] and full

security (FS) [13]. In UC, a general model protocol was proposed that can remain secure even if composed. The FS only allows access patterns to be leaked.

5) DYNAMIC IND-CKA2

For DSSE schemes, Kamara and Papamanthou [14] defined dynamic IND-CKA2 security, an extension of IND-CKA2 for update operation, which was stronger than the original IND-CKA2. Dynamic IND-CKA2 guarantees no leakage during an update operation (before any search operation is performed). Stefanov *et al.* [5] proposed forward and backward privacy (although no current DSSE schemes can achieve them), as discussed in Section 2(C).

6) DETERMINISTIC ENCRYPTION

If we use deterministic encryption to generate the ciphertext, then the ciphertext is the same every time for a given key and plaintext. To solve this problem, we can incorporate randomness to provide probabilistic encryption. For example, some SE schemes generate ciphertext with a random number. Although probabilistic encryption is more secure than deterministic encryption, deterministic encryption usually has higher efficiency.

E. MODEL

We describe the server and cryptography models for testing security.

1) UNTRUSTED SERVER MODE

The (untrusted) server model is honest-but-curious. It does not have any malicious action, but attempts to learn information about the plaintext.

2) RANDOM ORACLE AND STANDARD MODELS

SE schemes can be tested in random oracle (ROM) or standard (STM) models. STM assumes only computational complexity is required to provide security, because the adversary has limited resources, whereas ROM replaces cryptographic primitives by ideal versions, for example random oracle. Although DSSEs tested under ROM are often more efficient than under STM, it requires additional assumptions for the ideal cryptographic primitives.

III. RELATED WORKS

Here, we discuss the research results of safe searching, editing, and transmitting data in the application field of IoT. With the rapid development and growth of the IoT and cloud computing, a higher number of operators are outsourcing their IoT data to cloud servers to save costs or facilitate collaboration. Ma *et al.* designed an IoT public key encryption scheme that supports multiple keywords. This scheme supports database search, which does not require a secure network channel and is certificateless. [15].

Today, a large amount of data is outsourced from the IoT industry to the computing and storage resources of the cloud server. Because the devices provided by the cloud operators and their users are often not in the same trust domain, users cannot trust the security of the cloud server. Gao *et al.* proposed an M-SSE scheme based on multi-cloud scenario to achieve forward and backward security [16].

To deploy the SE protocol in the IoT environment, two aspects should be considered. 1. Security requirements: It is necessary to design methods to defend against common inside keyword guessing attacks and file injection attacks. 2. Computational overheads: The amount of data generated by IoT devices in some applications can be large. Efficient methods should be designed to address this. Wu *et al.* proposed an SE protocol that combines efficiency and security. The technology they used is a trapdoor replacement function [17].

The private information retrieval (PIR) protocol has been in existence for more than two decades. The main goal of PIR is to allow users to retrieve files from the server that owns the database but does not allow the cloud storage provider to know which file is being used. Riad *et al.* proposed an encrypted PIR scheme, which stores data on multiple different servers, each server does not store the entire database. It allows users to retrieve data without revealing user identity. [18].

In the previous paragraphs, a number of storage and search technologies based on the IoT application environment were introduced. These technologies are not resource-constrained for users. However, how to provide data storage and sharing services for mobile devices with limited resources is also crucial. Several studies have been conducted to perform simultaneous search queries on mobile devices and fine-grained access control for encrypted files in the database. For this topic, Miao *et al.* used a 0-encoding and 1-encoding technique to design a multi-keyword search scheme that can support comparable attributes [19].

The development of cloud computing and the Internet has resulted in increased popularity of the IoT industry, which has a considerable effect on the healthcare industry, leading to the transformation of the entire healthcare industry, allowing patients to participate more in the medical process and reducing the cost of the healthcare industry. IoT reduces the risk in the healthcare industry and reduces medical risks. This shift has ushered benefits and new opportunities to medical institutions at all levels, but it has also resulted in many security and privacy issues. Therefore, in the IoT application of the health care industry, searching and editing in the database of storing encrypted data through SSE technology is also a noteworthy trend. But for e-health applications, static SSE solutions that do not allow users to constantly update data are not recommended. Even if a user wants to use the existing DSSE solution, most of these DSSE solutions require processing for the first time in a large static data set, and then storing the data in the cloud device. This approach does not update the data frequently, which is completely different from the needs of the IoT-based e-health industry. The data of the patient may be required to be updated frequently. For example, constantly monitoring blood pressure data and

periodically creating and uploading it to the cloud database at a fixed frequency.

Therefore, Yang *et al.* proposed a reliable solution for the cloud-assisted body area network, which can search for data in the encrypted database without leaking sensitive data. This solution is a new DSSE system that, in addition to protecting privacy, also satisfies delegate verifiability, enabling patients and healthcare professionals to securely search and update encrypted data stored in the cloud and verify retrieved results [20].

Ocansey *et al.* proposed a DSSE scheme with forward privacy. This scheme requires that the server's database have an increasing counter, and must have a Bloom filter. The user must store state information on their own device. Then, using cryptographic primitives and an aggregated message authentication code, the scheme can implement forward privacy and allow multiple users to access files at the same time. This scheme generates tokens for use in search operations on the server, and is secure against malicious attack [21].

Next, we discuss studies on various DSSE schemes that were used. We explain the main concepts of each scheme and analyze and compare them according to five parameters: efficacy, security, client requirements, advantages, and drawbacks.

A. KPR DSSE SCHEME

The KPR DSSE scheme is proposed by Kamara *et al.* [3]. It is an extension to CGK-1 [22], based on invert indexes. The authors require any practical DSSE scheme to have the following properties: (1) sub-linear search time, (2) IND-CKA2 security, (3) compact index and add or delete files efficiently. KPR is the first DSSE scheme that satisfied all of these properties.

The main concept is to use two arrays to store the indexes. One is the invert index, called the search array, A_s; and the other is the forward index for deleting files, called the deletion array, A_d. Each invert index for a keyword (with encrypted file ID) is processed into a link-list and stored as A_s with random order. Each forward index for a file (with encrypted keyword) is similarly processed and stored as A_d, along with other encryption information, including the link-list relation in A_s. Thus, we can delete the encryption index in A_s, but not break the link-list because we can get the link information from A_d. We use two hash tables to point to the first entry in each array: Search table, T_s, points to the first entry for each keyword in A_s and deletion table, T_d, points to the first entry for each file ID in A_d.

However, this reveals the encrypted keywords contained in the file during the update process. For efficiency, the two tables and arrays need to be constructed in an interleaved mode, using pseudo-random random functions (PRF) to encrypt the tables and XOR to encrypt the arrays. Then we can use the reflexive relation for XOR $((A \oplus B) \oplus B = A \oplus 0 = A)$ update the arrays.

- Efficiency: KPR achieves optimal search time, and can update efficiently. It uses a hash table to search the first array entry and highly efficient XOR operation to encrypt and update array nodes. Keyed hash functions and PRFs are used to build the encrypted index.
- Security: KPR satisfies general IND-CKA2 security but leaks some information during the update process. KPR achieves security under ROM.
- Client requirement: client can prepare forward and invert index for building search and deletion arrays.
- Advantages: KPR uses invert index, achieving optimal search time, and the reflexive relation for XOR to update encrypted data efficiently. Drawbacks: XOR encryption leaks additional information during file updates and the data structure must be built in interleaved mode.
- Drawbacks: XOR encryption will leak additional information during file updates and the data structure must be built in interleaved mode.

Figure 3 shows the KPR scheme data structure and Figure 4 provides more details of the KPR scheme. Symbols k1 and k2 are keys for F and G. Note that A_d is used to record the linked relation in A_s, but this is not shown in the figure for simplification. Random numbers are used to increase randomness for encryption.

B. SPS SCHEME

The SPS scheme is proposed by Stefanov *et al.* [5]. It provides practical DSSE with a small leakage, and currently is the most secure DSSE scheme, leaking the least information to the server. And the server can access any element in constant time. SPS achieves not only IND-CKA2 security but also forward privacy through the hierarchical structure and special access method. SPS update protocol is interactive between the user and the server. When the user wants to add a new keyword-file pair (opcode can be add or del), the server returns the empty level and all encrypted data below the empty level. Then the usert decrypts the data from the server and re-encrypts with a different level key, i.e., the key of the empty level. The user then uploads the re-encrypted data and new encrypted keyword pair to the server. Thus, SPS achieves forward privacy since each element is re-encrypted with a fresh key and moved to a higher level. However, SPS has high computation and communication overheads, which is the first DSSE scheme to support dynamic keywords.

- Efficiency: SPS has sublinear search time in the worst case. However, N keyword-file pairs require $2N$ space. Encryption uses PRF, keyed hash function (random oracle), randomized symmetric encryption scheme (e.g. AES), standard hash function (e.g. SHA256), and XOR operations.
- Security: SPS achieves IND-CKA2 security and forward privacy. Leakages contain only the search, access, and file (the number of the keyword-file pairs) patterns. SPS achieves security under ROM.
- Client requirement: SPS only requires the forward index to build the encrypted index.

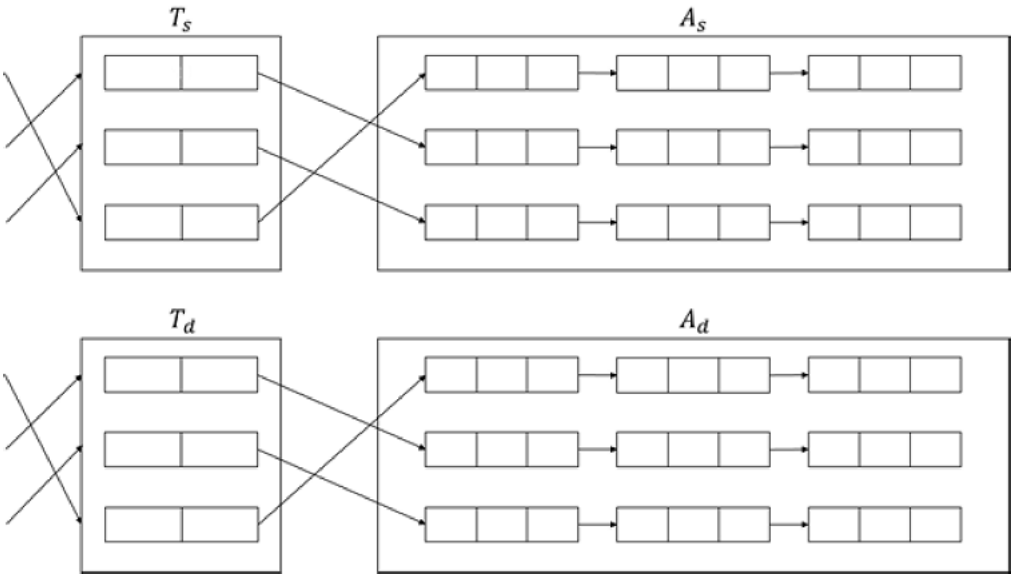


FIGURE 3. The sketchy data structure of KPR scheme.

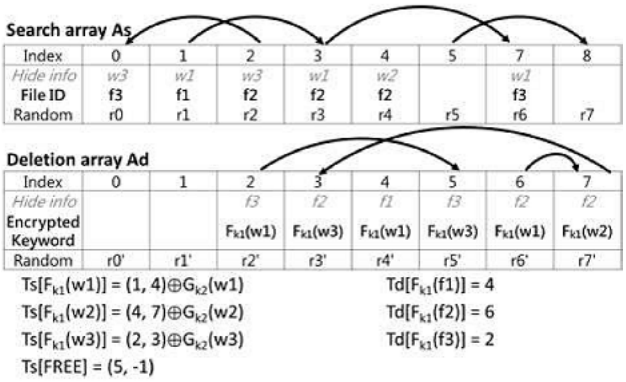


FIGURE 4. More detail about KPR scheme, where F and G are PRFs.

- Advantages: Leakages are currently the least, and the method can achieve forward privacy and support dynamic keywords.
- Drawbacks: SPS requires $2N$ space for N keyword-file pairs. For a single keyword search, the user generates more than one token, that is, one token per level. This has high overhead for update operations because every node is re-encrypted when moved to a higher level.

C. NPG SCHEME

The NPG DSSE scheme is proposed by Naveed *et al.* [6]. It uses the blind storage (BS) approach. The server is only required to provide storage, computation is executed by the user. Thus, the server is not required to perform any decryption, and thus achieves IND

- Efficiency: NPG has sublinear search time because it uses an invert index. However, updating overhead is high because to update an index file for a keyword, the whole index file should be retrieved. NPG cryptographic tools are block cipher, collision resistant hash function, and XOR operations. Although the server is not required to provide computation, server storage and communication costs are high.
- Security: Because the servers do not execute any decryption, NPG achieves IND-CKA2 security, and does not reveal the number of files or the size of each file until the files are accessed. However, it still leaks location information during a search query, because the user always accesses the same location for the same keyword. NPG achieves security under STM.
- Client requirement: NPG treats the invert index as a file set and uploads it to the server.
- Advantages: The server only provides cloud space, and hence can be built on many cloud services, such as Dropbox and Google drive. The BS scheme hides the actual operation (write, read, delete, and update). Search index files do not need a special format.
- Drawbacks: The user is required to download and decrypt superfluous blocks for an index file in BS for checking. Thus, NPG has high communication and update overheads.

We have not shown the BS structure in a figure, because it is just an array data structure.

D. HK SCHEME

The HK scheme is proposed by Hahn and Kerschbaum [7]. It provides SE with secure and efficient updates using a special construction. HK is based on the concept of learning the index from the access pattern to speed up search query for keywords searched often. The server maintains the (encrypted) forward index and the invert index simultaneously and the user maintains the search history. When the

user wants to add a new file, they generate a random number and a token for each keyword included in the file. Then, they generate ciphertext by a keyed hash function, using the random number as input and the token as key. Finally, the user sends the ciphertext and random number to the server as a forward index (indexed by the file ID directly).

HK is the first DSSE scheme with information leakage is than the access pattern, and keywords not yet searched do not reveal information.

- Efficiency: HK has sublinear search time by maintaining two indexes. It uses PRF, pseudo-random number generator, and keyed hash function (random oracle). It only requires two hash values for a keyword in a file.
- Security: HK satisfies IND-CKA2 security under ROM. Updation leakage is less than the access pattern.
- Client requirement: HK scheme clients prepare the forward index and maintain the search history.
- Advantages: Common keywords can be searched swiftly. Drawbacks: file deletion requires looking through the whole invert index.
- Drawbacks: File deletion requires looking through the whole invert index.

E. CJJJ SCHEME

The CJJJ scheme is proposed by Cash [4]. It provides DSSE for large databases. CJJJ uses a simple method to improve efficiency that allows large data sets. It combines several file identifiers and packs them into one ciphertext, and also has a two-level data structure. This structure reduces disk access and encryption times when building and searching. Thus, efficiency is high.

- Efficiency: CJJJ has high efficiency even if the dataset is large.
- Security: CJJJ satisfies IND-CKA2 security under ROM.
- Client requirement: CJJJ clients use the invert index to build and forward index to add.
- Advantages: High efficiency.
- Drawbacks: To classify the scale for each keyword, the client is required to compute the number of files that contain this keyword.

IV. METHODOLOGY

By weighing security and efficiency, we mix the various DSSEs and proposed a solution that hides the search pattern of the user. If forwarding privacy is to be implemented, then it will cause high overhead. Therefore, we allow the server to maintain both forward and invert index without using a hierarchy. The insignificant keyword method and the BS were used to store the inverted index to prevent the attacker from detecting the search pattern of the user.

A. SEARCH PATTERN HIDING

The search pattern is information that can allow the server to derive whether two search queries use the same keyword. We can use a non-deterministic algorithm to generate tokens and hide the search pattern, but it is difficult to design a DSSE scheme with non-deterministic search tokens.

A distributed searchable symmetric encryption scheme has been proposed recently [23], which can hide the search pattern. We treat the server as two parts: storage provider (SP) and query proxy (QP). The user and query proxies maintain independent search dictionaries. To search, the user generates two search tokens, T_{SP} and T_{QP}, for SP and QP, respectively. Then, SP re-encrypts and permutes the index according to T_{SP}, and QP can compute the new index location from T_{QP}. Thus, the user can generate different search tokens for a given keyword and hide the search pattern. However, this is only valid with the particular assumption that QP and SP do not collude, which seems unreasonable under the honest-but-curious server model. If the SP and QP collude, they are still honest-but-curious, because they do not break the protocol.

The search pattern will only be leaked when the keyword is searched more than once. Therefore, if we only allow the keyword to be searched once, we can hide the search pattern. Take achieve this, we propose a very simple concept to hide the search pattern. On receiving a search query, the server searches the encrypted index, returns the result to the user, and deletes the current encrypted index. When the user receives the search result, they update the index using an insignificant keyword. The insignificant keyword can be treat as an encrypted keyword or gibberish, but may also be plaintext, generated by XOR operation. The user needs to maintain the search history with search times, and records the new key generated to encrypt the keyword using XOR. The server supports dynamic keywords because the new keyword will be generated after search operation.

This is a new DSSE solution based on distributed computing. This solution is efficient because it only uses efficient primitives. In addition, in order to reduce the complexity of the database search, the inverted index technique, which may be vulnerable to attacks illegally accessing the search pattern, is also used. However, many previous studies have developed techniques dealing with oblivious RAM. In conjunction with the technique for dealing with private information retrieval problems, Bosch *et al.* proposed techniques for continually rearranging indexes while accessing them to avoid leaking the search pattern.

Since this technique uses the inverted index, it is necessary to create different indexes for each of the different keywords in order to search the encrypted database. The index is arranged according to each keyword of each document, and each bit maps to one of the above keywords.

Let the number of documents in the database be n. For each keyword w, make its plaintext index a string called i_w, and $|i_w| = n$. $i_w[j]$ will only correspond to a unique document, if the jth documents contain the keyword w, and then $i_w[j] = 1$, otherwise $i_w[j] = 0$. For the sake of confidentiality, iw will use XOR and some pseudo-random functions for encryption. The keys used for encryption are k_c, which is only known by the user, and k_s, which is known to both the user and the database. In general, k_c contains k_s. In the scheme

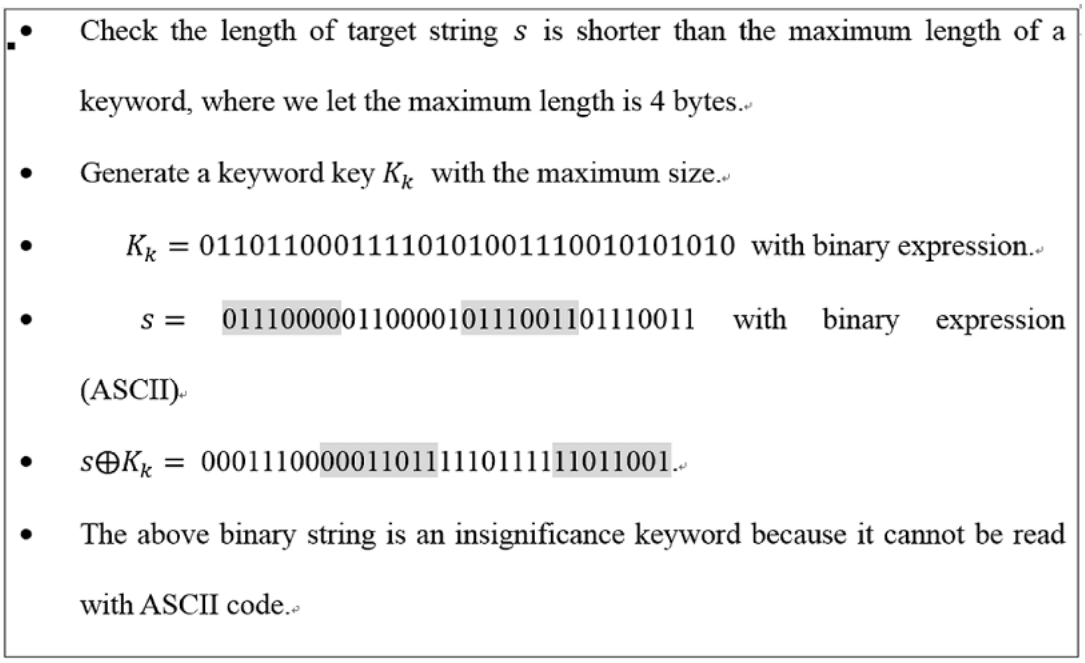


FIGURE 5. An example to generate an insignificance keyword.

proposed by Bosch *et al.*, the key used by the query proxy is empty. The encryption of the file is performed by the user using the symmetric encryption algorithm, and they do not use k_c as the key.

1) INSIGNIFICANT KEYWORD

Consider a string s = ''*pass*'', we transform s to an insignificant keyword following the steps in Figure 5.

To generate an insignificant keyword, a binary string containing any bytes that cannot be read in ASCII should be generated. The probability of successfully generating an insignificant keyword is related to the maximum length.

However, as the dictionary is fixed during the DSSE initialization, the current DSSE does not support the keyword updates; i.e., DSSE cannot search for the keywords not included in the dictionary, and cannot make searchable keywords unsearchable, which could be of practical interest to many applications. This study therefore proposes a DSSE with keyword updates. The rationale behind the proposed DSSE is the combined use of the existing DSSE and a variant of secure indexes. While the former offers functionality of the main DSSE, the latter is useful to the server in keeping track of undocumented and removed keywords.

In particular, with DSSE, the server receiving the search trapdoor from the user with a private key can perform a keyword search of encrypted documents without learning any part of the document content. Moreover, the server receiving the add-doc trapdoor (del-doc trapdoor) will accept new documents and corresponding keywords (remove the documents).

A dictionary consisting of all possible keywords is fixed during the DSSE initialization. While they support document updates, no current DSSEs support keyword updates. Here, keyword updates can be two operations. The first is to enable the data owner to search for a keyword currently not in the dictionary, while the second is to make a keyword no longer searchable by the data owner. Both of these operations can be achieved by either expanding or shrinking the dictionary.

An application scenario where the DSSE with keyword updates would be useful is shown in Figure 6. In the scenario, the management authority of an organization outsources encrypted documents to the server. To allow staff members to selectively retrieve documents, a key proxy with a DSSE key is deployed. If a staff member wants to search for a keyword, the keyword is sent to the key proxy via Intranet. The key proxy then transforms the keyword in plaintext into the keyword in DSSE format, and sends it to the server. Here, only the management authority with keyword-update key k_U can perform keyword updates, whereas the key proxy with only the DSSE key cannot. In this way, the authority is able to control which keywords can or cannot be searched by staff members. Unfortunately, though this application would be very useful, no current DSSEs support keyword update.

This paragraph describes the DSSE with keyword updates in this study. The idea behind the proposed DSSE can be thought of as a combined use of the existing DSSE [3] and a secure index variant [24]. More precisely, the existing DSSE is used for the regular DSSE operations, but in addition to the DSSE index, for each document the data owner constructs a secure index that includes all the single words that are in the document but are not in the current dictionary. Note that since DSSE [3] and secure index [24] both construct indexes, the indexes in [3] and in [24] are called the DSSE index and the secure index, respectively. In essence, the secure

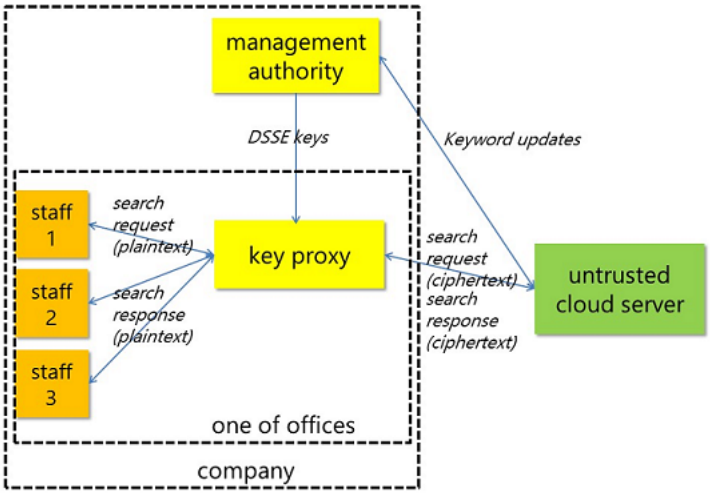


FIGURE 6. A conceptual use case of DSSE with keyword updates.

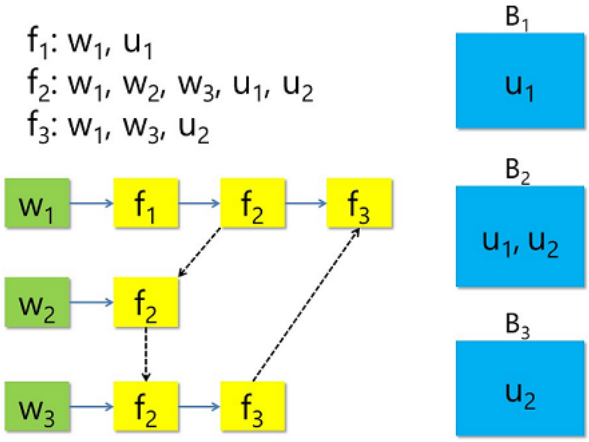


FIGURE 7. A DSSE index.

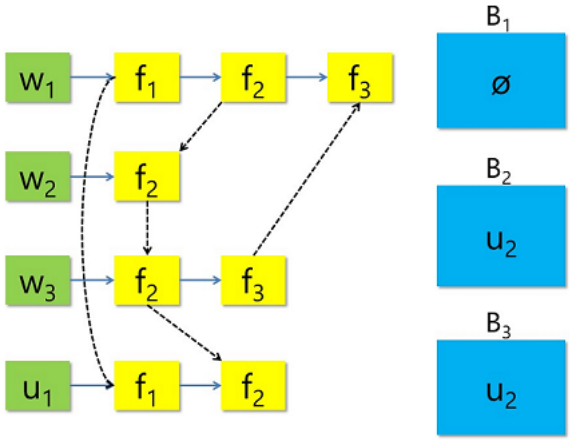


FIGURE 8. A DSSE index after add-keyword processing.

index for each document contains all keywords not in the current dictionary. Once index construction is complete, the DSSE and secure indexes are sent to the server. To process the add-keyword trapdoor, the server for each secure index checks whether the new keyword appears, and then modifies the DSSE index accordingly. To process the del-keyword trapdoor, the server modifies the DSSE index first, and then inserts the deleted keyword into the proper secure indexes. An illustrative example of how the proposed DSSE works is given in Figures 7, 8 and 9. More specifically, consider three documents f_1, f_2 and f_3, and a dictionary consisting of keywords w_1, w_2 and w_3. Document f_1 contains two words, one (w_1) is in the dictionary, and the other (u_1) is not. This study constructs a DSSE index in Figure 7 according to the method in [3]. The rationale behind the DSSE index in Figure 7 is to construct obscured linked lists based on keywords (blue arrows) (called keyword-based linked lists hereafter), and to construct obscured linked lists based on documents (black dash arrows) (called document-based linked lists hereafter). With these two categories of linked lists, the server can perform document updates. In addition, a secure index Bi is constructed for each document f_i. The secure index Bi is initialized as an empty counting Bloom filter [9]. All of the words in fi but not in the dictionary are added to B_i, for all $1 < i < n$, where n denotes the number of searchable documents.

B. HYBRID DSSE WITH SEARCH PATTERN HIDING

To design a practical DSSE scheme with search pattern hiding, we modified the models in [5] and [6] and combined them to propose a new hybrid DSSE (HDSSE) [7]. And limit the maximum keyword length cannot exceed the keyword key to produce negligible keywords. The high-level structure of the HDSSE scheme we propose is present in Figure 10.

This study divides the area used to store the index in the server into two parts, and uses the blind storage structure in the part containing the inverted index.

The purpose of using a blind storage scheme is to allow the user to store a set of files on an untrusted remote server, but the server has no way of knowing how many files are stored,

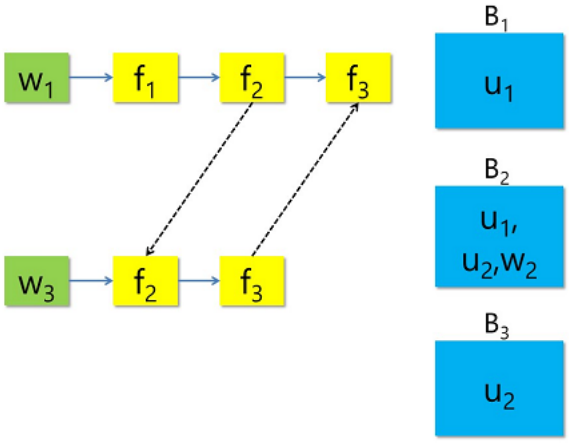


FIGURE 9. A DSSE index after del-keyword processing.

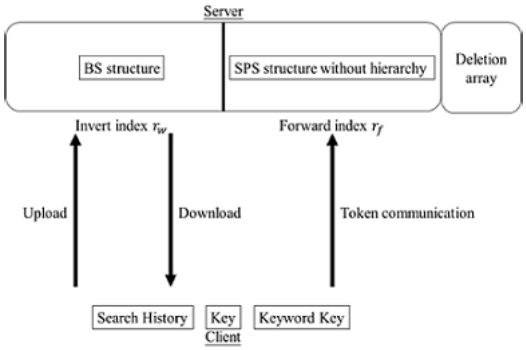


FIGURE 10. The high level for our hybrid DSSE scheme.

or the length of any single file. Although the server knows he file exists when the user retrieves it, and later knows that the user downloaded the file, the server cannot know the name of the file, and does not know the contents of the file. The blind storage scheme proposed by Naveed *et al.* also has the ability to add, update or delete files.

The blind storage scheme proposed by Naveed *et al.* is called SCATTERSTORE, and is very easy to use. This study uses the location generated by the pseudo-random function to store a file, and then treat the file as a set of blocks. The server only knows the superset of the file's location, not its exact location; each file only corresponds to a set of locations, and does not affect the files corresponding to the locations outside the set. This scheme requires only simple equipment, and is highly efficient in practical use.

The structure of the part of the storage containing the forward index emulates the structure proposed by Stefanov *et al.*, but without hierarchy levels in the proposed structure. This allows a user to release a token to search for keyword w. In this way, the server can decrypt all the data corresponding to the token, and can ensure that the sensitive data is known by the server, because the server cannot obtain any additional information during this process, and each token corresponds to a keyword.

In Stefanov *et al.*'s scheme, if a file needs to be added to the server, and it is known that the file contains keyword w, it is necessary to first determine how many files already contain w. Assuming that there are already $i-1$ files containing w, the hash function is used to encrypt w, x, add and i together. Similarly, if the user wants to delete the document, the hash function is used to encrypt w, x, del and I, and the encrypted result is stored. When the user wants to search for a document containing w, a token is generated to retrieve and decrypt the above stored data.

V. EXPERIMENT AND RESULT

We implemented each DSSE scheme discussed in Section 3 in C++ using the Crypto++ [25] open source library. We used AES to generate block ciphers and constructed a pseudo-random function [26]. The hash function was implemented using HMAC SHA256 and CMAC AES128. We used the Enron email dataset [27], as shown in Table 1 and Table 2, to test each DSSE. We stored each data structure on hard disk rather than in memory, because there was insufficient memory available on the test device to store that quantity of data. However, some DSSE schemes have very low hard drive efficiency, so we picked about 256 MB data from the Enron dataset to test build performance for each DSSE scheme. For add testing, we added two KB sized files; and for delete testing, we chose two files that were added at build time. For search testing, we searched ten frequently used words twice. The build dataset information is shown in Table 3 and Table 4.

TABLE 1. The information of Enron email dataset.

The number of file	517,375
The number of unique keyword	1,806,218
The number of file-keyword pairs	105,808,042

TABLE 2. The size information of Enron email dataset.

Full dataset	bytes	KB	MB
The sizer of file	1,421,103,627	1387796.51	1355.27
The size of list	663,832,713	648274.13	633.08
The size of index	1,186,632,993	1158821.28	1131.66

TABLE 3. The information of Enron email dataset for building test.

The number of file	79,986
The number of unique keyword	411,758
The number of file-keyword pairs	17,739,458

TABLE 4. The size information of Enron email dataset.

Full dataset	bytes	KB	MB
The sizer of file	267,809,999	261553.20	255.40
The size of list	112,464,559	109828.67	107.25
The size of index	199,431,709	194757.53	190.19

A. EXPERIMENT ENVIRONMENT

We run all programs as a single process on a laptop with i5-4210U CPU and 8 GB RAM. All data structures were

stored on the local solid state disk. First, we tested each DSSE scheme with a simple set, as shown in Table 2, and recorded build, search, add, and delete time. Then we tested each DSSE scheme with the 256 MB Enron email data subset, as shown in Error! Reference source not found.Table 3 and Table 4.

B. RESULTS

We measured performance using the number of pairs per second. For the search test, we also measured response time, because this is more intuitional.

1) BUILD TEST

The CJJJ scheme had very high performance due to the two-level structure and packing technique. When the dataset is medium, it packs several file identifiers to one ciphertext, stores them to dictionary, and uses pointers to retrieve these ciphertexts. The pointers are stored in an array. When the dataset is very large, the data is stored as in the medium size case, but the pointers are also packed and stored in an array. This structure reduces disk access and encryption times when building and searching, so has high efficiency.

Larger databases may result in tables that are too large, and the index will likely exceed the RAM capacity. The CJJJ method is not feasible in this case. Therefore, for each text column, the CJJJ scheme creates multiple tables. Tables cannot be too large, leaving enough of a RAM margin for the computer. The CJJJ scheme also requires that id-word pairs be evenly distributed across the table, however the same 'word' value must be in the same table. In addition, if necessary, the atomic column in the original table can also be similarly transformed.

As can be seen from the experiment results, the efficiency of the NPG scheme is also very good. The storage system designed consists of two main parts: the user, and the storage server. The function of the server itself is only downloading and uploading data, and the structure is simple, so it is very efficient when creating the system. In order to support the creation of the NPG scheme system, and to perform various database operations, the system requires several additional functions. The first is the key generating function. After inputting parameters, this function can generate keys corresponding to many cryptographic primitives simultaneously. The system is able to implement the NPG scheme in conjunction with the above key set and some parameters. These parameters include the upper bound of the amount of data stored, the files that the system initially stores, and the IDs of those files. Finally, the system also needs to establish the file access function. This function combines the above established information corresponding to the user's required operations, including uploading, downloading, updating and reading files. These operations should be able to interact correctly with the server and output the data required by the user. The contents of the file are placed in an encrypted data collection, and stored in a large array. The location file locations are provided by a pseudo-random number generator. These data and their interactions are stored in the server. As can be seen from the above, the process of establishing the system is not troublesome, so its efficiency is also good.

The performance of the HK scheme in the build test is similar to that of the NPG scheme, and it is faster than many DSSE schemes. The way to achieve this is to obtain the index from the search token. This saves time during system construction, and improves efficiency in the build test.

The principle of the HK scheme is to obtain the index in the access pattern. More specifically, the token and ciphertext are used to construct a list of stored indexes. However, this approach may require that the system operates for some time to achieve the desired search efficiency.

The number of HK scheme indexes has a linear relationship with the number of keywords. When constructing the system, the user can choose whether or not to store search history. This mainly refers to tokens used previously. This will directly affect the subsequent processing of the ciphertext to be stored. Saving a historical search record can improve the efficiency of the encryption in subsequent operations, but increase the time required to construct the HK scheme, because the server will create this record regardless of whether the user has established a history of the search.

SPS, the server data structure is hierarchical, with each level having a client generated key and 2^l elements to store encrypted data (keyword-file pairs), where $l = 0, 1, 2, \ldots, L$ is the level; and L is the maximum levels available in server, which is depend on the number of keyword-file pairs. The conceptual data structure includes the keywords, file IDs, opcode, and a counter for a given keyword and opcode, where opcode can be add or del, representing adding or deleting a keyword-file pair in the encrypted index, respectively. To speed access to each element in a level, a hash table is used to look up the elements, where the hash value is generated from the level key, keyword hash, opcode, and counter. This allows the server to access any element in constant time. We modified the building protocol to increase speed by computing the level number in advance, rather than updating every time. Figure 10 shows the main algorithm of SPS to encrypt data, and Figure 11 shows the conceptual unencrypted data structure.

$$\text{token}_l := F_{k_l}(\text{hash}(w))$$
$$\text{hkey} := H_{\text{token}_l}(0||\text{op}||\text{cnt})$$
$$\text{c1} := \text{id} \oplus H_{\text{token}_l}(1||\text{op}||\text{cnt})$$
$$\text{c2} := \text{Encrypt}_{\text{esk}}(w, \text{id}, \text{op}, \text{cnt})$$
$$\text{op} = \text{add or del}$$

FIGURE 11. Main algorithm for SPS scheme to store data to server.

Generally, the index-based SSE techniques use an encrypted inverted index, but the SPS scheme is different. The SPS scheme stores document-keyword pairs in a hierarchical structure, and the level of the hierarchical structure

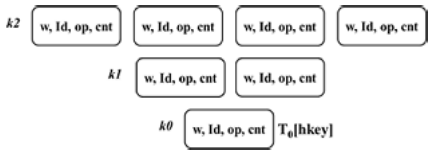


FIGURE 12. The conceptual unencrypted data structure on server for SPS scheme, where hkey is used to hash value to access in constant time.

is logarithmic. The SPS scheme uses the following method: if a document x contains keyword w to be added to the collection, and x is the ith document containing w, then the hash table is used to store the encrypted value of the tuple (w; x; add; i). Similarly, when a document x is removed from the collection, the tuple (w; x; del; i) is encrypted and stored. If a user wants to search for keyword w in the case of encryption, all the aforementioned hash table keys will be retrieved (and decrypted).

The proposed HDSSE system is a combination of [5], [6], and [7] so its performance is between them. The build performance of all systems is shown in Figure 13.

2) SEARCH TEST

We searched the same keyword twice to show the performance improvement after the first search for HK and the proposed HDSSE systems. In the HK scheme, the user sends the search token to the server, and the server uses the token as the key and the random number stored on forward index as input for the keyed hash function. If the ciphertext matches the output of keyed hash function, the server returns the file ID. The server also generates the invert index (indexed by the search token directly) and saves the search result. If the same token is queried by the user again, the server can look up the invert index directly and does not need to scan the whole forward index. Figure 14 and Figure 15 show the main idea for HK scheme.

HDSSE updates after every search to hide the search pattern, so the performance is lower than the other DSSE systems. The average response time for each DSSE scheme is shown in Figure 16 and the average performance of the search for each DSSE scheme is shown in Figure 17.

To implement the HK scheme, several data structures, such as lists and hash tables, are required. In order to search in the case where a file has been encrypted, the operations involved include the user performing the encryption operation, generating the token, and searching for the file in the server. However, it should be noted that the encrypted object is not limited to the file itself, but also includes encryption of keywords or other tools used to search for files. The encryption does not need to be performed by the user's device, as long as the storage server does not reveal the plaintext. The search token is also generated by the user who holds the key. The goal of this study is to have the server storing the files use this token to find all the corresponding ciphertexts, instead of just finding one.

In such a scheme, the primary concern is whether the operation mode described above can guarantee the security of the access pattern. In addition, the seriousness of the damage if the pattern is leaked must be determined. Since the user can search all the ciphertexts in one round of communication, a third party can obtain the access pattern via monitoring. Thus, in order to prevent the access pattern being leaked, some restrictions must be added, which will reduce performance quality and the practicality of the technique. Hahn *et al.* used the access pattern to create a list of indexes to implement the HK scheme. They encrypted and stored the keywords of each file. After searching for keywords, they put the file identifiers of all the files into the inverted index of the keyword, so that the keyword became the key of the index. The encrypted keyword then became a token for searching.

Of course, as the file changes, the keyword must be updated, especially those keywords that have already been searched. These keywords and the corresponding file identifiers are placed in the inverted index in the operations described above. Therefore, if the file is changed, the keyword must be corrected to the related inverted index, otherwise, there will be multiple indexes to be searched, resulting in lower performance. If the user can provide the server with keywords that have already been searched, the server can correctly maintain the inverted index, increasing efficiency.

It is safe to say that the performance of the CJJJ scheme in the search test is second only to that of the HK scheme. Although in the first search, CJJJ performed much better than the HK scheme. However, after the second search, the search results of the HK scheme are about six times that of the CJJJ scheme. On the other hand, the CJJJ scheme is far better than the other DSSE schemes.

The CJJJ scheme method is to generate an identifier for the stored file and the corresponding keyword, and create a dictionary to store these identifiers to assist the search operation. This method is actually not uncommon. When searching, the user enters the key and keyword, and the server outputs the identifier. When the identifier is stored, the server encrypts it, and the CJJJ scheme uses a specific counter to provide the corresponding label. The identifier can then be decrypted and output. In other words, in order to perform the search operation, in addition to matching the corresponding data, the server must compute the label for the keyword and decrypt the identifier stored in the dictionary. Since the file and the keyword are stored in pairs, the multiple searches for the same keyword will yield the same result. If a keyword corresponds to n files, then the CJJJ scheme will perform n searches in the dictionary to find the label to be provided. In the past, many DSSE schemes claimed to have good performance. The standard comparison method was to measure the number of files found in a specific time, but when the amount of data is very large, other factors must be considered. If the dictionary has to be stored in a secondary storage device that reads slower than the memory, the overall performance will be slowed down by each lookup in the secondary storage device. This problem does not only appear in the CJJJ scheme. Many schemes with

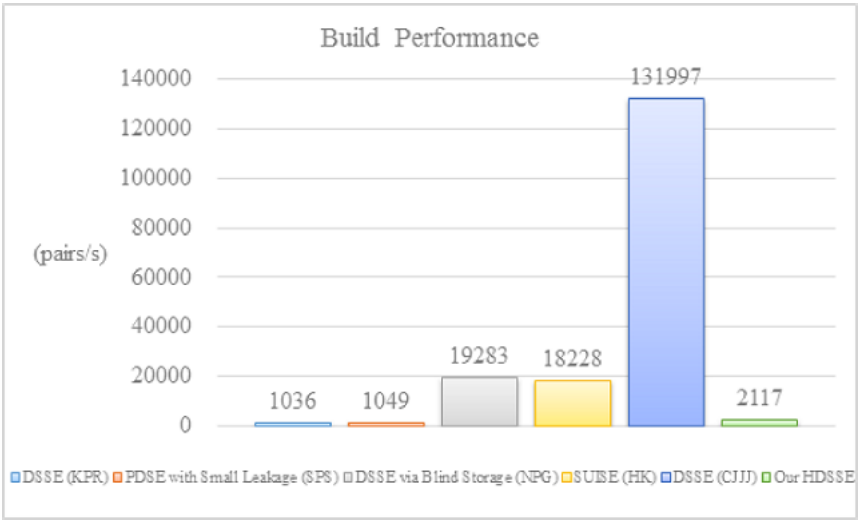


FIGURE 13. The performance of the first building for each DSSE scheme.

$$\text{token}_{w_i} := F_{k1}(w_i)$$
generate a random number s_i
$$c_i := H_{token_{w_i}}(s_i)||s_i$$

FIGURE 14. Main algorithm to generate encrypted forward index for HK scheme.

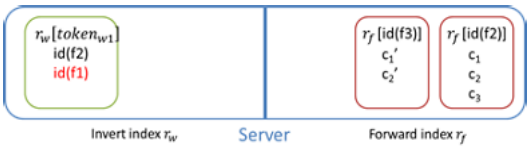


FIGURE 15. The two index maintained by the server in HK scheme.

similar practices have their search speed reduced by slow storage devices. Therefore, when dealing with a large amount of data, the CJJJ scheme replaces the dictionary with an array, improving efficiency.

3) ADD TEST

SPS has very low performance due to its level tree structure and high security. To search for a given keyword, the user generates search tokens for each level and forwards them to the server. The server uses the tokens to obtain the hash values for indexing the elements and finds the corresponding keyword pairs for both opcodes. The token allows partial decryption of each element to obtain the file ID containing the keyword and returns this to the user. Note that in the worst case (add k pairs and then delete $k-1$ pairs), SPS has lower efficiency because the server will search $2k+1$ times for a single result. Consequently, an extension keyword-file pair was proposed to record the corresponding add pair level. This ensured the search operation took sub-linear time.

In the NPG scheme, each file is divided into many blocks and stored scattered within an array. The array is encrypted and uploaded to cloud storage. Because the blocks are stored in pseudo-random locations within an array, where the pseudo random set is larger than the number of blocks for a file, the server cannot know the total number of files or the size of any file until it is accessed. When the user wants to retrieve a file from BS, they compute the pseudo-random location(s) for the file and download the corresponding blocks from the server, then decrypt them to access the file. The file can be updated by re-encryption and uploading. There is a very simple way to build DSSE via BS by treating the invert index for a keyword as a file and uploading all indexes to BS. We update the invert index for every keyword, so it also has low performance when adding new files. The performance of adding new file for each DSSE scheme is shown in Figure 18.

To implement the HK scheme, several data structures, such as lists and hash tables, are required. In order to search in the case where a file has been encrypted, the operations involved include the user performing the encryption operation, generating the token, and searching for the file in the server. However, it should be noted that the encrypted object is not limited to the file itself, but also includes encryption of keywords or other tools used to search for files. The encryption does not need to be performed by the user's device, as long as the storage server does not reveal the plaintext. The search token is also generated by the user who holds the key. The goal of this study is to have the server storing the files use this token to find all the corresponding ciphertexts, instead of just finding one.

In such a scheme, the primary concern is whether the operation mode described above can guarantee the security of the access pattern. In addition, the seriousness of the damage if the pattern is leaked must be determined. Since the user can search all the ciphertexts in one round of communication, a third party can obtain the access pattern

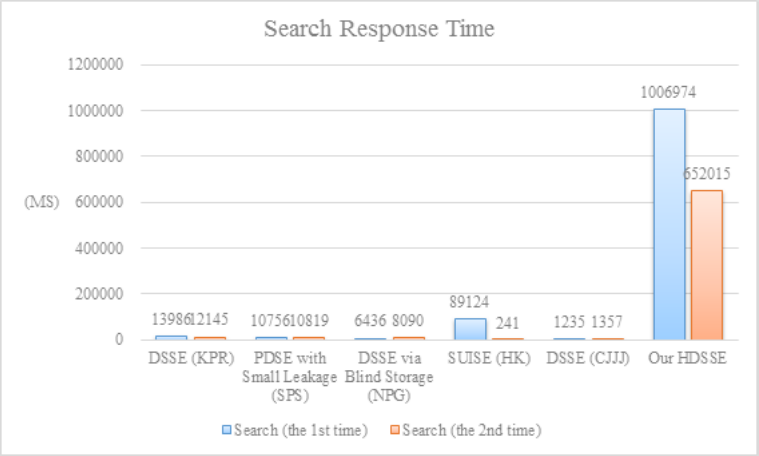


FIGURE 16. The average response time for each DSSE scheme.

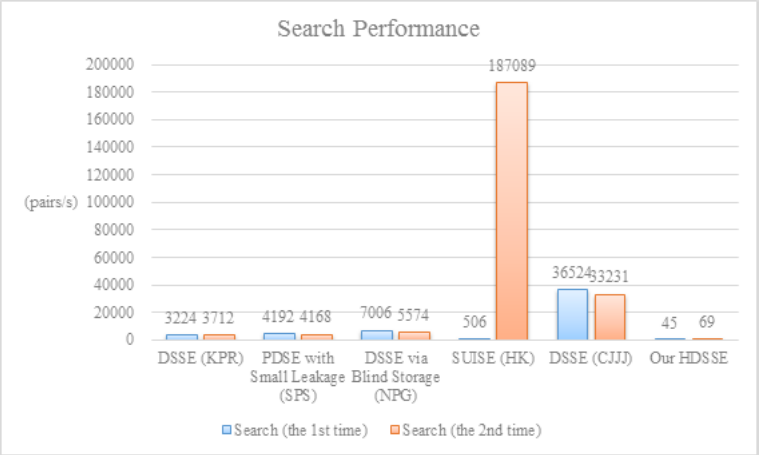


FIGURE 17. The average performance of the search for each DSSE scheme.

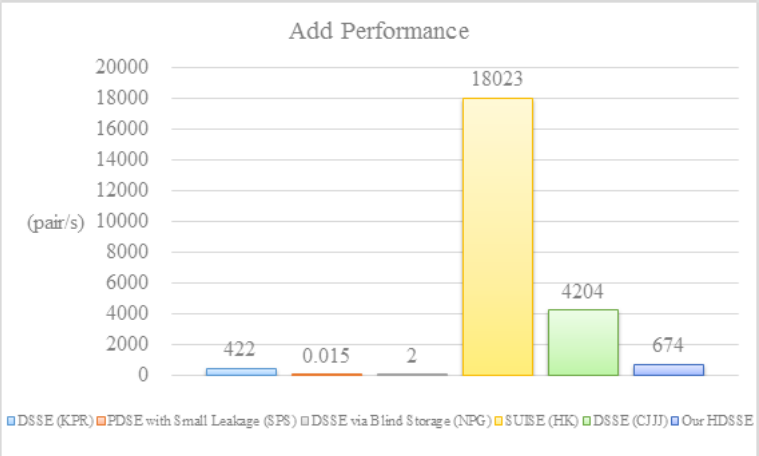


FIGURE 18. The performance of adding new file for each DSSE scheme.

via monitoring. Thus, in order to prevent the access pattern being leaked, some restrictions must be added, which will reduce performance quality and the practicality of the technique. Hahn *et al.* used the access pattern to create a list of indexes to implement the HK scheme. They encrypted and stored the keywords of each file. After searching for

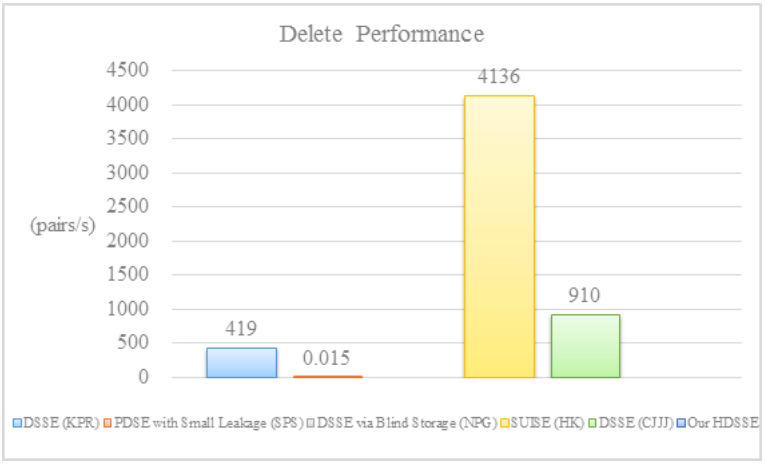


FIGURE 19. The performance of deleting for KPR, SPS, HK, and CJJ scheme.

keywords, they put the file identifiers of all the files into the inverted index of the keyword, so that the keyword became the key of the index. The encrypted keyword then became a token for searching.

Of course, as the file changes, the keyword must be updated, especially those keywords that have already been searched. These keywords and the corresponding file identifiers are placed in the inverted index in the operations described above. Therefore, if the file is changed, the keyword must be corrected to the related inverted index, otherwise, there will be multiple indexes to be searched, resulting in lower performance. If the user can provide the server with keywords that have already been searched, the server can correctly maintain the inverted index, increasing efficiency.

It is safe to say that the performance of the CJJJ scheme in the search test is second only to that of the HK scheme. Although in the first search, CJJJ performed much better than the HK scheme. However, after the second search, the search results of the HK scheme are about six times that of the CJJJ scheme. On the other hand, the CJJJ scheme is far better than the other DSSE schemes.

The CJJJ scheme method is to generate an identifier for the stored file and the corresponding keyword, and create a dictionary to store these identifiers to assist the search operation. This method is actually not uncommon. When searching, the user enters the key and keyword, and the server outputs the identifier. When the identifier is stored, the server encrypts it, and the CJJJ scheme uses a specific counter to provide the corresponding label. The identifier can then be decrypted and output. In other words, in order to perform the search operation, in addition to matching the corresponding data, the server must compute the label for the keyword and decrypt the identifier stored in the dictionary. Since the file and the keyword are stored in pairs, the multiple searches for the same keyword will yield the same result. If a keyword corresponds to n files, then the CJJJ scheme will perform n searches in the dictionary to find the label to be provided. In the past, many DSSE schemes claimed to have good performance. The standard comparison method was to measure the number of files found in a specific time, but when the amount of data is very large, other factors must be considered. If the dictionary has to be stored in a secondary storage device that reads slower than the memory, the overall performance will be slowed down by each lookup in the secondary storage device. This problem does not only appear in the CJJJ scheme. Many schemes with similar practices have their search speed reduced by slow storage devices. Therefore, when dealing with a large amount of data, the CJJJ scheme replaces the dictionary with an array, improving efficiency.

4) DELETE TEST

SPS has low performance because the data structure is exactly equal to the adding operation. We did not test delete performance for NPG and the proposed HDSSE systems because they delete files directly using the file system. The performance of deleting for KPR, SPS, HK, and CJJJ scheme is shown in Figure 19.

In terms of the deletion operation, it can be seen from Figure 15 that the HK scheme is the most efficient because when the HK scheme performs the delete operation, the user simply informs the server of the file identifier. The fewer steps required, the faster the performance will be. The HK scheme speed is second only to the CJJJ scheme. The CJJJ scheme creates a list on the server to help the server find the data to be deleted. Moreover, compared to other DSSE schemes, the CJJJ scheme does not reclaim the storage space immediately after deleting the data. Since the number of deletion operations is relatively small, the CJJJ scheme instead organizes the data on a regular basis instead of reclaiming the storage space after deletion. This also reduces the chance of revealing sensitive information. When a record and its corresponding keyword are deleted from the server, the user simply gives the server the corresponding revocation identifier. The process requires little time, as there are few steps involved.

As can be seen from Figure 15, the KPR scheme ranks third in terms of performance. This is related to the complexity

of the deletion process. In order to protect users' privacy, the DSSE scheme often encrypts related lists of various information. This makes it cumbersome to make any changes to the file. It is even difficult to determine to which node in the list the file being processed corresponds, because the data is already encrypted. Moreover, the chained list relies on the link of the pointer to maintain the correctness of the data. In the case of encryption, it is also difficult to maintain the correct pointer for each piece of data, and it may be difficult to identify available positions on the list.

In order to solve these problems, the KPR scheme adds a data structure that supports the delete operation. This design is rare in other DSSE schemes, but allows the server to handle the problem of modifying the list of pointers and finding the location of the node using a token and this extra data structure. When the user wants to delete a file, the server must delete its corresponding index, and then correct the data corresponding to the file in the keyword list. The server must use the delete token to find the record in the delete list, and then update the delete list and the various lists corresponding to the keyword. It should be noted that the token itself will not have the relevant information of the keyword. However, the server will know the identifier corresponding to the keyword in the process of deleting. In the server, each file has a corresponding node in the delete array and the search array, and in the two arrays, the nodes of the same file also correspond to each other. Therefore, when deleting a file, it is necessary to process the nodes in the two arrays at the same time. After the identifier receives the relevant information of the keyword, each time the server corrects the message of the deleted keyword, the location information of the keyword in the search array is also known. In this way, the server can maintain the correctness of the search array. However, these processes will inevitably have some impact on efficiency. In addition, the KPR scheme encrypts the pointer stored in each node in the array, but the server can modify the pointer without decrypting the node in the array. The encryption method used is a common XOR operation, so this encrypted action is still efficient.

VI. CONCLUSION

This paper compared the performance of seven DSSE schemes, including SPS, NPG, KPR, SPS, HK, CJJJ and the proposed DSSE scheme combining modified SPS, NPG, and HK, called HDSSE. From the experiment results, each scheme has its own advantages and disadvantages. If only speed is compared, then HK is very good for searching and adding and deleting files. However, when comparing security programs, sometimes other factors must be taken into account. Some users require a higher level of security to prevent illegal access to sensitive information. This is especially true in the IoT application environment, because IoT equipment collects extremely detailed information from users, and there could be a significant risk to user privacy if this information is not sufficiently secure. The proposed HDSSE solution therefore prevents access patterns being leaked. The proposed scheme thus requires a significant amount of time to search for data. However, the experiment also shows that although the proposed HDSE solution does not achieve the fastest implementation when adding files, it is only slower than HK and CJJJ. When deleting files, it only relies on the file system, and so requires little time.

ACKNOWLEDGMENT

The Enron email data used to support the findings of this study have been deposited in https://www.cs.cmu.edu/ ./enron/. This article was presented in part at the Institute of Electrical Engineering, National Taiwan University, 2015.

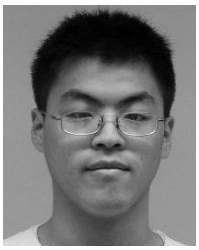
REFERENCES

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, May 2000, pp. 44–55.
[2] C. F. Wu, Y. W. Ti, S. Y. Kuo, and C. M. Yu, "Benchmarking dynamic searchable symmetric encryption with search pattern hiding," in *Proc. Int. Conf. Intell. Comput. Emerg. Appl.*, 2019.
[3] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 965–976.
[4] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, and M. Rosu, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2014.
[5] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2014.
[6] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2014, pp. 639–654.
[7] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 310–320.
[8] O. Goldreich, *Foundations of Cryptography: Basic Applications*, vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[9] E.-J. Goh, "Secure indexes," *Cryptol. ePrint Arch.*, vol. 2003, p. 216, 2003.
[10] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security*, vol. 3531, J. Ioannidis, A. Keromytis, and M. Yung, Eds. Berlin, Germany: Springer, 2005, pp. 442–455.
[11] 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.
[12] K. Kurosawa and Y. Ohtaki, "UC-secure searchable symmetric encryption," in *Financial Cryptography and Data Security*, vol. 7397, A. Keromytis, Ed. Berlin, Germany: Springer, 2012, pp. 285–298.
[13] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Advances in Cryptology—EUROCRYPT*, vol. 6110, H. Gilbert, Ed. Berlin, Germany: Springer-Verlag, 2010, pp. 62–91.
[14] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography and Data Security*. Springer, 2013, pp. 258–274.
[15] M. Ma, D. He, N. Kumar, K.-K. R. Choo, and J. Chen, "Certificateless searchable public key encryption scheme for industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 14, no. 2, pp. 759–767, Feb. 2018.
[16] C. Gao, S. Lv, Y. Wei, Z. Wang, Z. Liu, and X. Cheng, "M-SSE: An effective searchable symmetric encryption with enhanced security for mobile devices," *IEEE Access*, vol. 6, pp. 38860–38869, 2018.
[17] L. Wu, B. Chen, K.-K. R. Choo, and D. He, "Efficient and secure searchable encryption protocol for cloud-based Internet of Things," *J. Parallel Distrib. Comput.*, vol. 111, pp. 152–161, Jan. 2018.
[18] K. Riad and L. Ke, "Secure storage and retrieval of IoT data based on private information retrieval," *Wireless Commun. Mobile Comput.*, vol. 2018, 2018, Art. no. 5452463.

[19] Y. Miao, J. Ma, X. Liu, X. Li, Z. Liu, and H. Li, "Practical attribute-based multi-keyword search scheme in mobile crowdsourcing," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3008–3018, Dec. 2017.
[20] L. Yang, Q. Zheng, and X. Fan, "RSPP: A reliable, searchable and privacy-preserving e-healthcare system for cloud-assisted body area networks," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.
[21] S. K. Ocansey, W. Ametepe, X. W. Li, and C. Wang, "Dynamic searchable encryption with privacy protection for cloud computing," *Int. J. Commun. Syst.*, vol. 31, no. 1, pp. 1–8, 2018.
[22] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," presented at the 13th ACM Conf. Comput. Commun. Secur., Alexandria, VA, USA, 2006.
[23] C. Bosch, A. Peter, B. Leenders, L. Hoon Wei, T. Qiang, and W. Huaxiong, "Distributed searchable symmetric encryption," in *Proc. 25th Annu. Int. Conf. Privacy, Secur. Trust (PST)*, 2014, pp. 330–337.
[24] W. Dai. (Feb. 20, 2013). *Crypto++ Library 5.6.2*. [Online]. Available: http:/www.cryptopp.com
[25] L. Fan, P. Cao, J. Almeida, and A. Z. A. Broder, "Summary cache: Scalable wide-area Web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.
[26] J. Song, R. Poovendran, J. Lee, and T. Iwata, *The Advanced Encryption Standard-Cipher-Based Message Authentication Code-Pseudo-Random Function-128 (AES-CMAC-PRF-128) Algorithm for the Internet Key Exchange Protocol (IKE)*, document RFC 4615, Aug. 2006.
[27] W. W. Cohen. (2015). *Enron Email Dataset*. [Online]. Available: https://www.cs.cmu.edu/~/enron/

YEN-WU TI received the B.E. degree in mathematics from Tam-Kang University and the M.E. degree in applied mathematics from National Chiao-Tung University, Hsin-Chu, Taiwan, in 1995 and 1997, respectively, and the Ph.D. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 2009.

He is currently an Associate Professor with the College of Artificial Intelligence, Yango University, China. His research interests include algorithm and computation theory.



CHIA-FENG WU received the M.S. degree from National Taiwan University, in 2015. His research interest includes searchable encryption.

CHIA-MU YU received the Ph.D. degree from National Taiwan University, in 2012.

He was a Postdoctoral Researcher with the IBM Thomas J. Watson Research Center. He was a Visiting Scholar with Harvard University, Imperial College London, Waseda University, University of Padova, and University of Illinois at Chicago. He is currently an Assistant Professor with the National Chung Hsing University, Taiwan. His research interests include differentially private mechanism design, cloud storage security, and the IoT security. He received the K. T. Li Young Researcher Award from ACM/IICM, Observational Research Scholarship from Pan Wen Yuan Foundation, and the Project for Excellent Junior Research Investigators from the Ministry of Science and Technology, Taiwan. He has served as an Associate Editor of IEEE Access and the *Security and Communication Networks*.

SY-YEN KUO received the B.S. degree in electrical engineering from National Taiwan University (NTU), Taipei, Taiwan, in 1979, the M.S. degree in electrical and computer engineering from the University of California at Santa Barbara, in 1982, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, in 1987.

He was a Faculty Member of the Department of Electrical and Computer Engineering, University of Arizona, from 1988 to 1991, and an Engineer at Fairchild Semiconductor and Silvar-Lisco, both in CA, USA, from 1982 to 1984. In 1989, he also worked as a Summer Faculty Fellow with the Jet Propulsion Laboratory, California Institute of Technology. He spent his sabbatical years as a Visiting Professor with The Hong Kong Polytechnic University, from 2011 to 2012, and with The Chinese University of Hong Kong, from 2004 to 2005, as well as a Visiting Researcher at AT&T Labs-Research, Middletown, NJ, USA, from 1999 to 2000, respectively. He was the Dean of the College of Electrical Engineering and Computer Science, NTU, from 2012 to 2015, and the Chairman of the Department of Electrical Engineering, NTU, from 2001 to 2004. He also took a leave from NTU, and has served as a Chair Professor as well as the Dean of the College of Electrical Engineering and Computer Science, National Taiwan University of Science and Technology, from 2006 to 2009. He is currently the Pegatron Chair Professor with the Department of Electrical Engineering, NTU. His current research interests include dependable systems and networks, mobile computing, cloud computing, and quantum computing and communications.