IDCrypt: A Secure and Practical Searchable Encryption Scheme for Cloud Applications

Guofeng Wang [1], Chuanyi Liu [2*], Yingfei Dong [3], Peiyi Han [1], Hezhong Pan [1], Binxing Fang [2]

[1]School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China

[2]Harbin Institute of Technology (Shenzhen), Shenzhen, 518055, China

[3]Department of Electrical Engineering, University of Hawaii Honolulu 96822, Hawaii, USA

(* Corresponding Author: cy-liu04@mails.tsinghua.edu.cn)

***Abstract*—Searchable Encryption (SE) has been extensively examined by both academic and industry researchers. While many academic SE schemes show provable security, they usually expose some query information (e.g., search patterns) to achieve high efficiency. However, several inference attacks have exploited such leakage, e.g., a query recovery attack can convert opaque query trapdoors to their corresponding keywords based on some prior knowledge. On the other hand, many proposed SE schemes require significant modification of existing applications, which makes them less practical, weak in usability, and difficult to deploy. In this paper, we introduce *a secure and practical SE scheme with provable security strength* for cloud applications, called IDCrypt, which improves the search efficiency and enhanced the security strength of SE using symmetric cryptography. We further point out the main challenges in securely searching on *multiple indexes* and sharing encrypted data between *multiple users*. To address the above issues, we propose a token-adjustment scheme to preserve the search functionality among multi-indexes, and a key sharing scheme which combines Identity-Based Encryption (IBE) and Public-Key Encryption (PKE). Our experimental results show that the overhead of IDCrypt is fairly low.**

***Keywords*—Cloud security; Searchable encryption; Key sharing**

I. INTRODUCTION

According to Cloud Security Alliance (CSA) report [1], data breach is one of the top threats in cloud computing. In the first half of 2016, there were 974 publicly disclosed data breaches, which led to the loss of 554 million data records [2]. A promising solution to this issue is to encrypt sensitive data before outsourcing to a cloud service.

However, the encryption that protects user data hinders the search functionality of cloud applications. To address this issue, many Searchable Encryption (SE) schemes have been proposed, which usually consists of the following steps: A user generates its encrypted documents and a searchable encrypted index. The encrypted documents and the encrypted index are then outsourced to the cloud. To search a keyword, the user generates a so-called *trapdoor*. With the trapdoor, the cloud can search on the encrypted index and return related documents. With such steps, many proposed SE schemes require significant modifications of existing application and query interface customization, which makes them difficult to deploy in practice.

SE constructions usually try to exploit the trade-offs among functionality, security, and efficiency [3]. Song et al. [4] proposed the first practical SE construction. However, it shows a low efficiency and does not introduce a formal definition of SE security. Goh [5] proposed a secure index scheme that guaranteed the index security but not trapdoors; its search time is linear with the number of documents. Curtmola et al. [6] proposed the first sub-linear scheme which is more suitable for a *static* document set than a dynamic one. They introduced two new SE security definitions requiring that nothing should be leaked from the search process beyond the search and access patterns, which are widely used in current SE schemes [3]. As far as we know, there is no practical SE scheme that hides query access patterns. While the Oblivious RAM (ORAM) [7] can be used to support SE functions without leaking any information, it is not efficient for large-scale practical use.

In recent years, several inference attacks have been designed to exploit access pattern disclosures with prior knowledge. Islam et al. [8] proposed the first inference attack against SE schemes based on the leakage of access patterns. Cash et al. [9] exploited the leakage profiles of various SE schemes to infer sensitive information about queries and documents. However, these attacks require knowing almost all the documents as prior knowledge to achieve a high recovery rate of queries.

In this paper, we propose a secure and practical Symmetric Searchable Encryption (SSE) scheme, called IDCrypt, for protecting sensitive information in the cloud. In the IDCrypt architecture, we deploy an index server that can be configured using off-the-shelf index-based search engines [10-11] to provide search functions on encrypted documents, which

improves search efficiency and security strength of SE. IDCrypt achieves provable security and can mitigate the above inference attacks effectively. We further point out the main challenges in securely searching different indexes and sharing encrypted data among multi-users, summarized as follows:

(i) Searching encrypted data across different indexes. A scenario is that user u_1 (who always uses secure search function on its own index I_1) wants to search the encrypted data on another index I_2 that is encrypted by another user u_2.

(ii) Encrypted data sharing between multi-users. A typical scenario is that user u_1 wants to send an encrypted message to user u_2. How to securely share encrypted data and keys among different users is another difficult challenge.

To address above challenges, a token adjustment scheme is proposed in this paper to preserve the search functionality when a user searching on different indexes encrypted with different keys. Moreover, we design a *Two-Layer Encryption Scheme (TLES)*, which combines Identity-Based Encryption (IBE) and Public-Key Encryption (PKE), to securely share secret keys between different users. We have evaluated the proposed schemes, and our experimental results show that the extra overhead of IDCrypt is fairly low.

In summary, the main contributions of this paper are:

1) We proposed a practical symmetric SE scheme with provable security strength, called IDCrypt, to improve the search efficiency and enhance the security of SE. It achieves provable security and can effectively mitigate the inference attacks on SE. The SSE schemes on IDCrypt are compatible with off-the-shelf index-based search engines, which can be easily deployed in practice. They can also support advanced SE functions, including multi-keyword ranked SE and dynamic SE.
2) We propose a token-adjustment search scheme to preserve the search functionality when a user need to search indexes encrypted by different keys.
3) We design a Two-Layer Encryption Scheme (TLES), which combines Identity-Based Encryption (IBE) and Public-Key Encryption (PKE), in order to share secret keys between different users securely and efficiently.

The remainder of this paper is structured as follows. In Section 2, we discuss the security definitions of SE and inference attacks on SE schemes. In Section 3, we introduce our IDCrypt architecture and depict its security analysis in detail. We present the token-adjustment search scheme in Section 5, and develop the TLES scheme in Section 6. We show the implementation of IDCrypt and evaluate its performance in Section 7. Related works are summarized in Section 8. We further conclude this paper and discuss our future work in Section 9.

II. BACKGROUND

In this section, we introduce the basic concepts used in this paper, and present the background information of the security definitions of SE and inference attacks on SE schemes.

A. Terminology & Preliminary

n — The total number of documents in a collection
$D = \{D_1, D_2, \ldots, D_n\}$ — A collection of n documents
$ID(D_i)$ — The identifier of document D_i
$D(w)$ — The ordered list consisting of the identifiers of all documents in D that contain the keyword w
m — The total number of keywords in a dictionary
$W = \{w_1, w_2, \ldots, w_m\}$ — The set of keywords in a dictionary
Q — A series of queries, each represents a query q
I – The search index for document set D

Bilinear Map. Let G_1 and G_2 be two cyclic groups of order p, for some large prime p. G_1 is the group of points of an elliptic curve over $|F_q|$, and G_2 is a subgroup of $F^*_{q^2}$. A mapping $\hat{e}: G_1 \times G_1 \rightarrow G_2$ is said to be bilinear if $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ for all $P, Q \in G_1$ and all $a, b \in Z$.

Security of Pseudo-Random Function. A pseudo-random function $f: \{0,1\}^s \times \{0,1\}^n \rightarrow \{0,1\}^m$ is computable in polynomial time and for all probabilistic polynomial-time adversaries A, all polynomials p and sufficiently large s:

$$\left| Pr\left[\mathrm{A}^{f_k(\cdot)} = 1 : k \xleftarrow{R} \{0,1\}^s \right] - Pr\left[\mathrm{A}^{g(\cdot)} = 1 : g \xleftarrow{R} \{F : \{0,1\}^n \rightarrow \{0,1\}^m\} \right] \right| < \frac{1}{p(s)}$$

Security of Symmetric Encryption. A symmetric encryption scheme $E' = (K, E, D)$ is secure against chosen-ciphertext attacks if for all probabilistic polynomial-time adversaries A, all polynomials p and sufficiently large s:

$$Pr\left[b' = b : K \leftarrow (\mathscr{K}(1^s); (m_0, m_1) \leftarrow \mathrm{A}^{\mathscr{E}_K(\cdot), \mathscr{D}_K(\cdot);} \; b \xleftarrow{R} \{0,1\}; c \leftarrow \mathscr{E}_K(m_b); b' \leftarrow \mathrm{A}^{\mathscr{E}_K(\cdot), \mathscr{D}_K(\cdot)}(c)\right] < \frac{1}{p(s)}$$

with the restrictions that $|m_0| = |m_1|$, and A cannot query D with c.

Symmetric Searchable Encryption (SSE): A SSE scheme is a collection of four polynomial-time algorithms as follows:

Keygen(1^s): a probabilistic key generation algorithm that takes a security parameter s, and returns a secret key k. It is run by the user to setup the scheme.

BuildIndex(K, D): a (possibly probabilistic) algorithm takes a secret key k and a document collection D as inputs, and outputs an index I.

Trapdoor(k, w): an algorithm that takes a secret key k and a word w as inputs, and outputs a trapdoor T_w. It is run by the user to perform a search operation.

Search(I, T_w): an algorithm that takes an index I and a trapdoor T_w for word w as inputs, and returns a search result $D(w)$.

We refer to a sequence of query results $(D(w_1), \ldots, D(w_i))$ of a sequence of query keywords $(w_1, \ldots, w_i)$ as an access pattern. The search pattern means the information that can be derived in the following manner: given two arbitrary queries, knowing whether the two searches use the same keyword or not. Referring to [6][12], the security strength of common SSE schemes follows the real/ideal simulation paradigm. They leak certain information to achieve better performance and we

Commented [YD1]: define F_q
Commented [YD2]: F_q anf F_q* are not defined.
Commented [YD3]: E()'s are not defined
Commented [YD4]: s, n, and m are not defined
Commented [YD5]: S, n, m are not defined

quantify the leakage profile by a leakage function L. The leakage function takes a sequence of queries Q as input and outputs what an adversary learns by taking part in the execution of the SE scheme. The SSE scheme is said to be *L-secure* if there exists a simulator S, which takes an input $L(P)$ with P a history of the protocol and outputs a view $S(L(P))$ that is indistinguishable from the view of an adversary in a real execution of the protocol with input P.

Security of SSE. Let $SSE = (KeyGen, Enc, Trapdoor, Search, Dec)$ be a SSE scheme, A is a stateful Probabilistic Polynomial-Time (PPT) adversary, S is a stateful PPT simulator, L_1 and L_2 are stateful leakage functions in an ideal security game, and $s \in N$ denotes the security parameter. We define $Real_A(s)$ and $Ideal_{A,S}(s)$ games as follows:

$Real_A(s)$: the challenger runs $KeyGen(1^s)$ to generate a key K. A outputs D and receives $(I, C) \leftarrow Enc_K(D)$ from the challenger. Then the adversary makes a polynomial number of adaptive queries Q, and for each query q of keyword w, receives from the challenger a search trapdoor $TD \leftarrow Trapdoor_K(w)$, Finally, A returns a bit b that is output by the game.

$Ideal_{A,S}(s)$: A outputs D. Given $L_1(D)$, S generates and sends a pair (I, C) to A. The adversary makes a polynomial number of adaptive queries Q and, for each query q of keyword w, the simulator receives $L_2(D, w)$ and returns an appropriate trapdoor TD. Finally, A returns a bit b that is output by the game.

We say that SSE is (L_1, L_2) semantically secure against adaptive attacks if, for all (non-uniform) PPT adversaries A, all polynomials p and sufficiently large s, there exists a (non-uniform) PPT simulator S such that

$$|Pr[Real_A(s) = 1] - Pr[Ideal_{A,S}(s) = 1]| < 1/p(s)$$

We define (L_1, L_2) semantically secure against non-adaptive attacks in the same way, except that A must choose all of its queries at the start in both games. It can only provide security if the client's queries are independent of the search index and previous query results.

Common SSE models can be classified into token-based SE and index-based SE. Token-based SE scheme can search users' encrypted sensitive data without modifying cloud Application Programming Interface (API). As shown in Fig.1, a token-based SE scheme is depicted as follows.

1) A user extracts keywords from a document D_i, generates a token for each keyword by deterministically encrypted the keyword using a pseudo-random function f (such as HMAC) and a secret key k, and encrypts document D_i using a secret key K. Then the user appends the sorted tokens to the ciphertext, and uploads them to the cloud.
2) When searching for a keyword w, the broker applies the keyword to the pseudo-random function f and sends generated token $TK=f_k(w)$ to the server.
3) The server can search for the token TK using the original search algorithms and return the corresponding encrypted data or encrypted document identifiers.

With the above discussion, we can conclude that besides the sizes of the documents and tokens, the L_1 leakage in the token-based SSE model exposed the information of the number of keywords in a document, the document similarity, the keyword occurrence pattern and the keyword co-occurrence pattern. The L_2 leakage exposed the search pattern and the access pattern, which can be inferred from the L_1 leakage.

An Index-based SE scheme delegates search capabilities to a cloud provider on behalf of a user, which should modify cloud API to invoke specific SE libraries. Curtmola, et al. [6] first proposed an index-based SSE scheme that lets nothing but the search and access patterns be known to the server. For each keyword w, it built a posting list consisting of $|D(w)|$ nodes. A posting contains an identifier of a document containing w, a key used to decrypt the next encrypted posting, and a pointer to the next encrypted posting. All postings are encrypted with random keys and scrambled in a random order. As shown in Fig. 1, the index-based SE scheme contains the following important steps:

1) A user generates a searchable encrypted index I and the encrypted documents $E_K(D)$ independently, and uploads the encrypted index I and the encrypted data $E_K(D)$ to the cloud.
2) Then, the user generates the trapdoor $TD = f_k(w)$ of a keyword w for search.
3) With the trapdoor TD, the cloud service can search on the encrypted index using specific search algorithms and return the corresponding encrypted data or encrypted document identifiers.

Based on the security definitions of the symmetric encryption E and the pseudo-random function f, we conclude that the L_1 leakage in the index-based SSE model exposed nothing besides the sizes of the documents and indexes. However, the L_2 leakage exposed the information of search patterns and access patterns. From this leakage, we can infer keyword occurrence patterns and keyword co-occurrence patterns for queried keywords.

Inference Attacks: Inference attacks on SE schemes often use the information of keyword occurrence frequency and multi-keyword co-occurrence frequency to guess the query keywords or document contents. They usually contain the following steps.

1. An adversary used the leakage information of the SE scheme (e.g., search patterns and access patterns) to infer the frequency of the query keyword or co-occurrence frequency of multiple query keywords.
2. With prior knowledge of the documents, the adversary can count the frequency of plaintext keyword and the co-occurrence frequency of multi-keywords respectively.
3. With the plaintext statistics and the query information, the adversary designed matching algorithms to invert the query keywords.

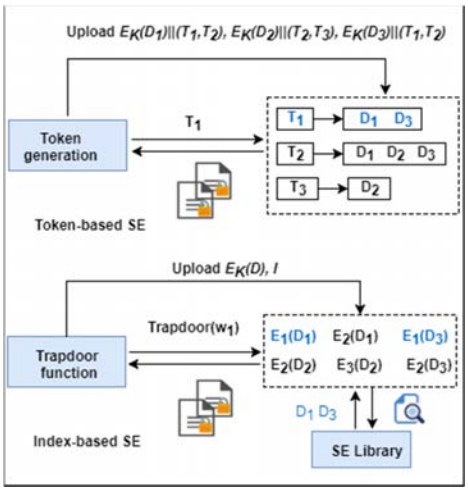


Fig. 1. Searchable encryption system models.

Islam, Kuzu, and Kantarcioglu (IKK) [8] first initiated the inference attacks on SE with the help of access pattern disclosure. With sufficient prior knowledge, they convert their attack model as an optimization problem and uses simulated annealing [13] to find the optimal mapping between query trapdoors and keywords. Cash, Grubbs, Perry, and Ristenpart (CGPR) [9] proposed a simple and efficient attack algorithm to address the same issue as IKK. A Token-based SSE scheme encrypted each keyword using deterministic encryption algorithms, which leaks token occurrence patterns. From the leakage, an adversary can obtain the keyword occurrence frequency and multi-keyword co-occurrence frequency in the document set. Based on this, Shadow Nemesis [14] launched inference attacks on the token-based SE schemes using the Weighted Graph Matching (WGM) problem. However, to invert an encrypted keyword with a high accuracy, an adversary must have sufficient knowledge of the documents in the above attacks.

III. IDCrypt Architecture

We first provide a high-level system overview of our IDCrypt architecture in the following, and then propose two easily-deployable SSE schemes that improve the efficiency and security of SE with provable security definitions.

A. System overview

IDCrypt encrypts the confidential data of a user using a proxy before outsourcing it to cloud servers. To support encrypted search functions, the proxy of IDCrypt on the client side indexes the document data with the document identifier that points to the encrypted data in the cloud, and relies on an index server to assist in managing and searching on the index, thus taking the management burden off the proxy of IDCrypt. In particular, the IDCrypt paradigm focuses on usability and does not need to modify the cloud API and a user's familiar usage patterns. As shown in Fig. 2, IDCrypt consists of the following main components:

- Proxy. A proxy is responsible for encrypting a user's confidential data. It is also responsible for generating the searchable encrypted index for a document set with document identifiers that point to the encrypted documents on the cloud.
- Index server. An index server is responsible for managing the index data and executing the search operations.

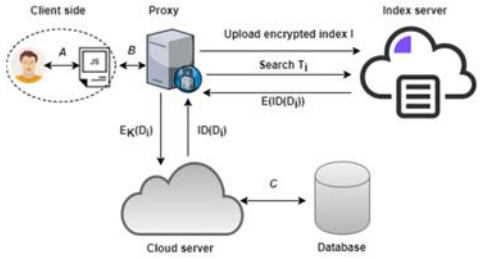


Fig. 2. IDCrypt architecture.

IDCrypt assumes that the processing in a proxy is trustworthy and secure, namely the data within the proxy is transmitted in clear texts. Besides, since IDCrypt indexes the plaintext data inside the proxy with identifiers pointing to encrypted data at cloud servers, it has to protect the proxy from external accesses and attacks.

The pathways outside a user's proxy, including the cloud servers, the index server and other proxies are considered untrusted and need to be protected. As shown in Fig. 2, the sensitive data is encrypted by the proxy before passed to the outside, thus effectively preventing the cloud servers and the index server from stealing private information. In addition, even if the cloud servers and the index server collude, they cannot recover the plaintexts because they cannot get the corresponding keys that are located in the user's proxy. Outside the proxy, even if the user's account is stolen by an attacker, the attacker can only access encrypted data, because the encrypted data does not go through the proxy and there is no decryption process.

Commented [YD6]: becuase the encrypted data must be decrypted by the proxy server. (I am not clear what you want to say here.)

B. The SSE-I scheme

To construct an efficient inverted index over the keywords in a document set, IDCrypt locates the index server at an online third-party service provider. IDCrypt encrypts the searchable index to prevent the third-party provider's snooping, and protect privacy information (e.g., searchable keywords and document identifiers), as shown in Algorithm 1. The procedures of the SSE-I scheme are as follows:

1) For each document in a document set, the proxy first extracts keywords from the document, and generates searchable tokens by deterministically encrypting the extracted keywords.

3) Then, the proxy gets the document identifiers that point to the encrypted documents in the cloud. It maps the tokens of a document with corresponding document identifiers respectively.

3) Finally, the proxy indexes the searchable tokens with their *encrypted* document identifiers, and uploads the encrypted index to the index server.

4) When a user performs a search operation for a keyword, the proxy sends the corresponding token of the keyword to the index server; the index server performs the search operation and returns the encrypted document identifiers.

5) After decrypted by the proxy, the document identifiers can be used to request the encrypted documents from the cloud servers.

The index structure of the SSE-I scheme is shown in Fig. 3. For each keyword $w \in W$, a linked posting list $L(t)$ with length $|D(w)|$ is built, in which t is the corresponding token of keyword w. Each item of $L(t)$ consists of an encrypted document identifier $ID(D_i)$ of D_i that contains the keyword w. The entries of a look-up table W_t are tuples <value, address>, in which the value field is used to locate the corresponding posting list by a pseudo-random function H. That is, when a keyword is queried, we use its corresponding token value to locate its posting list.

When a user initiates a query request, the proxy analyzes the request to get the query keyword. The proxy then generates the corresponding token of the keyword and sends a request with the token to the index server. With the token, the index server searches on the index and returns the corresponding encrypted document identifiers. After decrypting the encrypted document identifiers, the proxy can request cloud servers for corresponding encrypted documents. When receiving the data, the proxy decrypts it and returns the plaintext data to the end user. In this way, the SSE-I scheme hides the access pattern to a certain extent since document identifiers are encrypted, such that an adversary can not deduce the keyword co-occurrence frequency from the query results. However, it leaks the number of documents in which a keyword appears. We emphasize that the sizes of documents and the index will also be leaked, which is common and not mentioned in our security defenations.

Algorithm 1: SSE-I algorithms

Algorithm *Keygen* (1^s)
  Given a security parameter s, output keys K, K', k.
Algorithm *BuildIndex* (K, D)
  Compute $E_K(D)$ and upload it to the cloud;
  for ***each document*** D_i *in* D **do**
    get the document identifier $ID(D_i)$ of D_i;
    for ***each keyword*** w_i *in* D_i **do**
      token $t_i = f_k(w_i)$)
    build the mapping between the tokens $T_i = \{t_1, ...t_i...\}$ and $ID(D_i)$;
    for ***each*** t *in* T_i **do**
      if t *in dictionary* W_t **then**
        get $L(t)$ according to $H(t)$, add $E_{K'}(ID(D_i))$ to $L(t)$;
      else
        add t to W_t;
        initialize a posting list $L(t)$ according to $H(t)$, add $E_{K'}(ID(D_i))$ to $L(t)$.
  output index $I = (W_t, L)$.
Algorithm *Trapdoor* (w)
  Output $TK_w = f_k(w)$.
Algorithm *Search* (I, TK_w)
  Search TK_w in the index I.;
  Output the search result $E_{K'}(D(w))$.

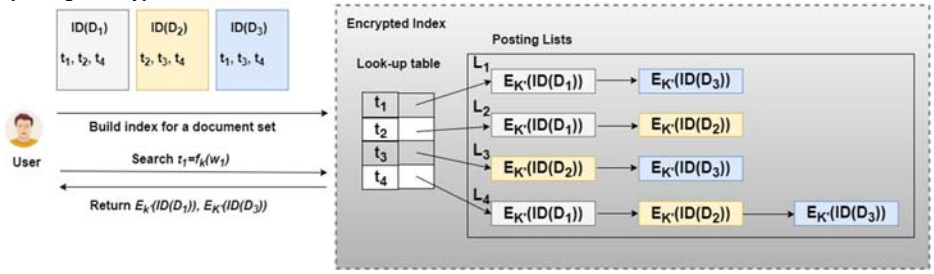


Fig. 3. Index structure of the SSE-I scheme.

Theorem 3.1. The SSE-I scheme is adaptively (L_1, L_2) secure, in which the L_1 leakage exposed the number of searchable tokens, the occurrence count of each keyword (the number of encrypted document identifiers of each token). The L_2 leakage exposed search patterns, and the occurrence counts of query keywords from access patterns, which can be

deduced from L_1 leakage.

Proof: For the proof, we follow the security definition of SSE [6][12]. We say our SSE-I scheme is (L_1, L_2) semantically secure against adaptive attacks if, for all (non-uniform) PPT adversaries A, all polynomials p and sufficiently large s, there exists a (non-uniform) PPT simulator S that can simulate the encrypted index I and ciphertexts $E_K(D)$ with a probability negligibly close to 1.

At a high level, the PPT simulator S builds a simulated encrypted index I^* and a simulated sequence of ciphertexts $E^*_K(D)$ using the information that it receives from L_1 leakage, which includes the number of searchable tokens, and the occurrence count of each keyword. The ideal index I^* can be constructed similarly to a real index, except that the encrypted document identifiers are replaced by random strings and the output tokens of the pseudo-random functions are replaced by random values.

Generating posting lists L^*: For each keyword w and its occurrence count $|D(w)|$, S generates $|D(w)|$ random strings and sets them as a posting list for w.

Generating look-up table W^*: For each keyword w, S generates a random string t and lets $H(t)$ point to the posting list of w. S records the correspondence between w and t.

Generating query trapdoor T^*: For the query keyword w, S sets its trapdoor T^* as t corresponds to the keyword w.

The security of the symmetric encryption and the security of pseudo-random function guarantee that the resulting encrypted index $I^* = \{W^*, L^*\}$ is indistinguishable from a real encrypted index, and the resulting trapdoor T^* is indistinguishable from a real trapdoor.

The simulated document encryptions $E^*_K(D)$ are simulated in the same manner (i.e., replacing ciphertexts by random strings) and the security of the symmetric encryption guarantees indistinguishability. This completes the proof of the theorem. ■

Our SSE-I scheme is very efficient, in which the time complexity of index building is $O(N)$, where $N = \sum_w |D(w)|$, the time complexity of searching is $O(1)$, and the index size is $o(m+n)$, where m is the total number of keywords, n is the total number of documents. Besides, the scheme hides keyword co-occurrence patterns, as all document identifiers are encrypted to random values. However, the scheme exposed the number of documents in which a keyword appears, which may be exploited by an adversary to infer some queries if the adversary has sufficient knowledge of the documents. In the following, we design a scheme called SSE-II that can hide keyword occurrence frequencies.

C. SSE-II scheme

In our SSE-II scheme, we pad the length of the posting list of each keyword to a fixed number c. We can determine the number c according to the most frequent keyword in a document set. Before building the index for a document set, we need to count the occurrence number of the most frequent keyword. An alternative way is setting the number c as the total number of documents. This method is more efficient at the cost of more space. After determining the number c, for each keyword, if its occurrence number is less than c, we pad the length of its posting list to c with random strings. Our *BuildIndex* algorithm of SSE-II scheme is shown in Algorithm 2, and the *Keygen*, *Trapdoor* and *Search* algorithms are the same with the ones in SSE-I scheme.

Theorem 3.2. The SSE-II scheme is adaptively (L_1, L_2) secure, where L_1 leakage exposes the number of searchable tokens, and L_2 leakage exposes the search pattern.

The proof of this theorem is similar to that of Theorem 3.1; except that in the process of generating posting lists, for each keyword w, the simulator S generates c random strings and sets them as a posting list for w.

In our SSE-II scheme the time complexity of index building is $O(cn)$, the time complexity of searching is $O(1)$, and the index size is $o(m+cn)$, where m is the total number of keywords, and n is the total number of documents. Besides, the scheme hides the number of documents in which a keyword appears and the keyword co-occurrence patterns, as all document identifiers are encrypted to random values. So our SSE-II scheme can mitigate the inference attacks on SE even if an adversary has complete knowledge of the documents. Note that after receiving the search result of a keyword, the proxy should filter the padded random strings after decryption in the SSE-II scheme.

Algorithm 2: SSE-II algorithms

Algorithm *BuildIndex (K, D)*
- Compute $E_K(D)$ and upload it to the cloud;
- Determine the fixed number c according to D;
- **for** ***each document*** D_i *in* D **do**
  - get the document identifier $ID(D_i)$ of D_i;
  - **for** ***each keyword*** w_i *in* D_i **do**
    - token $t_i = f_k(w_i)$
  - build the mapping between the tokens $T_i = \{t_1, ...t_i...\}$ and $ID(D_i)$;
  - **for** ***each*** t *in* T_i **do**
    - **if** t *in dictionary* W_t **then**
      - get $L(t)$ according to $H(t)$, add $E_{K'}(ID(D_i))$ to $L(t)$;
    - **else**
      - add t to W_t;
      - initialize a posting list $L(t)$ according to $H(t)$, add $E_{K'}(ID(D_i))$ to $L(t)$.
- **for** ***each token*** t *in* W_t **do**
  - Pad the length of the posting list of t to c.
- output index $I = (W_t, L)$.

D. Advanced Searchable Encryption Functions

In this section, we discuss the advanced search functions [3][15] that can be implemented efficiently in our SSE schemes, including multi-keyword ranked SE and dynamic SE.

Multi-keyword ranked SE: In a multi-keyword ranked SE, a user wants to seek k documents with the highest relevance scores on the query vector Q from a document set D. To rank the documents which contain more than one keyword of the query vector Q, we use the following equation to compute the relevance scores of matched documents, in which $TF(u, w_i)$ means the normalized Term Frequency (TF) value of keyword

Commented [YD7]: the page number is wrong since page 6. Please fix.

w_i in D_u, and $IDF(w_i)$ represents the normalized Inverse Document Frequency (*IDF*) value of keyword w_i in D.

$\boldsymbol{Score(D_u, Q)}$. The function calculates the relevance score between query vector Q and a document D_u.

$$Score(D_u, Q) = D_u \cdot Q = \sum_{w_i \in W_q} TF_{u,w_i} \times IDF_{w_i}$$

In the index building process, we need to add a *TF* value in each item of a posting list to represent the frequency of a keyword in a document. Besides, we compute an *IDF* value of each distinct keyword in all the documents, and attach it to the corresponding token of the look-up table. When performing a multi-keyword ranked query, we execute the following procedures:

1. For each keyword in the query Q, the index server returns all the encrypted identifiers in its posting list.
2. On receiving the return results, a proxy decrypts them to get all the related document identifiers and corresponding *TF/IDF* values.
3. If a query is an "or" query, we compute the score of each document identifier according to Equation 1 and return the document identifiers having the top k scores; If the query is an "and" query, we first select the documents which contain all the keywords in the query, and then compute the scores of selected documents.

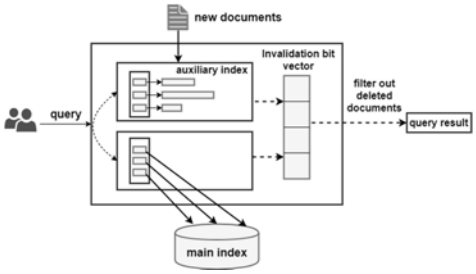
Dynamic SE: According to Kamara et al. [12], a practical SE scheme should be dynamic. It can support update operations that add files to an existing index or delete files from the index. In a dynamic SE scheme, new terms can be added to a dictionary of an index, and the posting lists of the index can be updated for existing terms.

For our SSE schemes, an inverted index is built as shown in Fig. 4. However, performing an in-place update on the main inverted index in the index server directly is time-consuming [15]. To add new documents quickly, we maintain two indexes: a large main index stored in the index server and a small auxiliary index that points to new documents and is kept in the proxy. When the auxiliary index is larger than a threshold, it is merged into the main index in the index server. In the SSE-II scheme, when merging the auxiliary index into the main index, we also need to pad the lengths of posting lists in the auxiliary index to a fixed number (the occurrence number of the most frequent keyword or the number of documents). There are several typical operations in our dynamic SSE schemes.

- Search: The operations are processed on both the main index and the auxiliary index, and then the results are merged.
- Add: When adding a new document, the proxy will build the auxiliary index for the document.
- Delete: When a document is deleted, an invalidation bit vector stored in the proxy will be updated to indicate that the document has been deleted. Then the proxy will filter out the deleted documents according to the vector before returning the search result.
- Update: If a document is updated, it will be deleted and re-inserted.

When a large proportion of documents have been deleted from a document set, we need to generate a new index of the remaining documents to replace the old index for saving storage space. The off-line re-index process is depicted as follows:

1. For each token of an index, the proxy initiates a query to the index server with the token.
2. After receiving and decrypting the search results, the proxy validates each document identifier against the invalidation bit vector stored in the proxy, and removes the document identifiers that have been deleted.
3. The building of the new index is completed when all the search tokens in the old index have been processed, and all the invalid document identifiers have been deleted from the posting lists. In the SSE-II scheme, we also need to pad the length of each posting list in the new index to a fixed number.
4. Finally, the proxy encrypts and uploads the new index to the index server, and removes the old index. Then subsequent queries are performed on the new index instead of the old index.



- Fig. 4 Dynamic search in our SSE scheme

IV. Encrypted Search on Multi-Indexes

In practice, a user may have different document sets in different cloud applications, so the indexes of different document sets may be encrypted using different keys. When a user wants to search for all document sets, he must generate multiple distinct search tokens for a keyword, and send all of them to the index server. In another case, there are multi-users in a cloud application. The documents of different users and their indexes are encrypted with different keys. Many efficient SSE schemes [4-6] assume that a search trapdoor is encrypted using a single key, and they are suitable for a single-user scenario. If a user u_1 wants to delegate another user to search its documents for a keyword, one approach is that user u_1 sends the request instead of the other user, and then sends the result to the other user. However, when the same request is repeated many times by other users, this approach is inefficient for user u_1. Another alternative method is exposing the key of user u_1 to other users, but this approach allows other users to search for the keywords other than they requested. To address this issue, we design a token-adjustment search scheme, presented in the following.

A. Token-adjustment search

In our token-adjustment search scheme, if a user wants to search for a keyword in a set of indexes, in which each index is encrypted with a different key, the user only needs to provide a single search token for that keyword to the index server. The index server, in turn, returns each encrypted identifier of a document that contains the keyword.

We assume user Alice's own key is k_1, and the index I that Alice searches for is encrypted with key k_2 by user Bob. Alice computes a search token for a keyword w using key k_1, denoted t_1. If the corresponding token of w of the index I in the index server is t_2 instead of t_1, the index server must adjust the search token t_1 to t_2.

The token-adjustment search scheme allows the index server to perform the adjustment. In the first place, the index owner must provide a delta, which is a cryptographic value that enables the index server to adjust a search token from one key to another key. We use $\Delta_{t1\to t2}$ to denote the delta that allows the index server to adjust t_1 to t_2. Note that these deltas can be reused for subsequent search operations, so the index owner needs to generate the deltas only once. For example, if Bob delegates a search right of a keyword to Alice, he needs to provide one delta to the index server, such that the index server will be able to adjust the token t_1 from Alice to a searchable token t_2 using the token adjustment search scheme.

B. Cryptographic construction

We construct the token adjustment search scheme based on the key derivation algorithm proposed by Atallah et al [16], in which a delta between two keys $k_1$and k_2 is defined as follows:

$$\Delta_{ki\to kj} = k_i \oplus f(k_i, L_j)$$

where L_j is a public label associated with k_i, $\oplus$ is the XOR operator and f is a pseudo-random function. Referring to Equation 2, our token-adjustment search algorithm is depicted as follows:

Token-adjustment search algorithm. Given two tokens t_i and t_j that are encrypted by different keys for keyword w, a delta between t_i and t_j is defined as $\Delta_{ti\to tj} = t_j \oplus f(t_i, L_j)$, where L_j is a public label associated with the keyword w, $\oplus$ is the XOR operator, and f is a pseudorandom function, which can be implemented using a cryptographic hash function such as HMAC.

Based on the token-adjustment search algorithm, the procedure of the token-adjustment search scheme is depicted as follows:

1. If Alice wants to search a keyword w on the index I built by Bob, she sends a request to Bob with the value $f(t_i, L_j)$, in which t_i is the token of w computed by Alice, L_j is a publicly available label associated with the keyword w.

2. On receiving the request, Bob checks whether Alice has the right to search for w. If passed, Bob computes the delta value $\Delta_{ti\to tj} = t_j \oplus f(t_i, L_j)$ with the corresponding token t_j of w. Finally, it uploads the delta to the index server.

3. In the subsequent searches, if Alice wants to search for the keyword w, she can send the value t_i to the index server, then the index server can adjust the value to t_j by computing $f(t_i, L_j)$, then obtaining $t_j = \Delta_{ti\to tj} \oplus f(t_i, L_j) = t_j \oplus f(t_i, L_j) \oplus f(t_i, L_j)$, thus the index server can search the token t_j on the index I built by Bob.

For a user, different keywords of his documents have different labels; for a common keyword, different users have different tokens encrypted with different keys. In this way, a user can control which keyword can be searched by which user. So our token-adjustment search scheme can realize fine-grained access control.

V. Encrypted Data Sharing Between Proxies

The data is encrypted at the proxy, namely the secret key is located in the proxy. So only the proxy can decrypt the encrypted data. A typical scenario is that user u_1, on premise of proxy P_1, wants to share a file with user u_2, who is under another proxy P_2. When user u_2 receives the encrypted file from the cloud, it is encrypted by proxy P_1. So the problem is how to share encrypted data between different proxies.

To address the above issue, secret key sharing is an obvious solution. However, the secret key must be shared securely, and the receiver must provide a credential that is trusted by both parties. To do this, an IBE scheme [17] can achieve implicit certification, but has a private-key escrow problem, namely the Private Key Generator (PKG) can decrypt the encrypted data of the user. Alternatively, a PKE scheme does not have a private key escrow issue, but is inefficient in the case of revoking lots of certificates or bringing many third-party queries for certificate status [18]. We combine IBE and PKE to realize a Two-Layer Encryption Scheme (TLES), which adopts their unique advantages to address their deficiencies. Finally, we design a practical prototype system and perform performance evaluations.

To realize TLES, we deploy a control node acts as the PKG of an IBE scheme. When a proxy is initialized, the proxy authenticates to the control node using its own PKE public key and its identity ID. After authentication, the control node issues an IBE private key corresponding to the identity ID. The control node is also responsible for initializing and updating the basic information of a proxy, such as proxy ID, proxy public key, etc. A proxy can query some related attributions (e.g., proxy ID and proxy public key) of another proxy from the control node.

When a data block is encrypted at a proxy, a metadata, including key ID, proxy ID, magic data, header length, etc., is attached to the encrypted data. A *magic tag* is a symbol string used to identify the encrypted data, so the decryption process can find the ciphertext easily. The proxy records the key ID that marks the relationship between the encrypted data and its corresponding key. The decryption process first locates the ciphertext according to the magic tag in the metadata. It then obtains the corresponding key according to the key ID and decrypts the ciphertext to restore plaintext data. Let the identity ID of the proxy denote its IBE publics key, d denote the IBE private key, PK denote the PKE public key, and SK denote the PKE private key. As shown in Fig. 5, the procedure of TLES scheme between different proxies is presented as follows:

1-3) Proxy *B* receives the ciphertext encrypted by proxy *A* from the cloud.

4-8) Proxy *B* obtains the metadata of the ciphertext. After getting some necessary information from the control node, it finds Proxy *A* according to proxy ID in the metadata and then requests the key from proxy *A* with parameters. The parameters include the key ID and (possibly) the proxy *B*'s time parameter t_B.

9) After obtaining some information about Proxy *B* if necessary, Proxy *A* double encrypts the requested key with parameters t_B, proxy *B*'s identity ID_B, broker *B*'s PKI public key PK_B, and forwards the encrypted key to proxy *B*.

10) Proxy *B* uses the IBE private key d_B and the PKI private key SK_B to decrypt received messages and obtains the corresponding key. Then the proxy *B* decrypts the ciphertext with the decrypted key to obtain plaintext data.

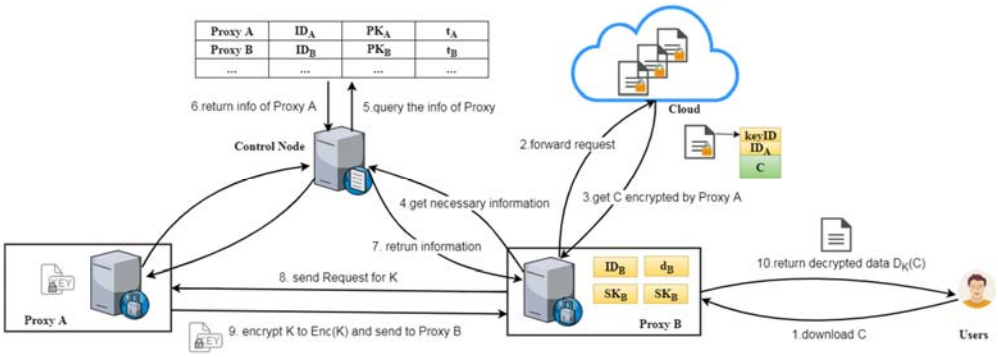


Fig. 5. Encrypted data sharing between proxies.

Referring to [17], our concrete TLES algorithm based on the Bilinear Map is shown as follows:

Setup: The control node generates the system parameters $params = < q, n, P, P_{pub}, G, H >$and master-key s. After setup, the control node exports *params* to a file and forwards it to proxies.

- Step 1: The pairings are constructed on the curve $y^2 = x^3 + x$ over the field F_q for a prime $q = 3\ (mod\ 4)$. We generate prime q by the formula $q + 1 = t * h$, in which t is a prime and h is a multiple of 12. For efficiency, t has the form $2^a - 2^b - 1$ for integer a and b, $0 < b < a$. For security strength, we set the length of $t = 160$ bits, the length of $q = 512$ bits. Let E be the elliptic curve defined by $y^2 = x^3 + x$ over F_q. Choose an arbitrary $P \in E / F_q$.
- Step 2: Pick a random $s \in Z_q^*$ and set $P_{pub} = sP$.
- Step 3: Choose a cryptographic hash function $H: F_{q^2} \rightarrow \{0,1\}^n$ for n. Choose a cryptographic hash function $G: \{0,1\}^* \rightarrow Fq$. The security analysis will view H and G as random oracles. The message space is $\{0,1\}^n$. The ciphertext space is $E / F_q \times \{0,1\}^n$.

Extract: For a given string $ID \in \{0,1\}^*$ from the proxy identity and time parameters, generate a private key d as follows:

- Step 1: Map ID to a point $Q_{ID} \in E / F_q$.
- Step 2: Set the private key d_{ID} to be $d_{ID} = sQ_{ID}$, where s is the master key.

The control node then encrypts the private key d using proxy's PKE public key and forwards it to the proxy. The proxy decrypts it to get the private key *d*.

Encrypt: Encrypt $M \in \{0,1\}^n$ under the public key ID and public key PK_B:

- Step 1: Map ID into a point Q_{ID},
- Step 2: choose a random $r \in Z_q$, and
- Step 3: set $C_1 = < rP, M \oplus H(g_{ID}^r) >$, where $g_{ID} = \hat{e}(Q_{ID}, P_{pub}) \in F_q$,
- Step 4: Encrypts rP with the broker PKE public key PK_B, set the ciphertext to be $C = < E_{PK_B}(rP), M \oplus H(g_{ID}^r) >$.

Decrypt: Decrypt M with the PKI private key SK_B and the private key *d*:

- Step 1: Let $C = < E_{PK_B}(rP), M \oplus H(g_{ID}^r) >$, Decrypts $E_{PK_B}(rP)$ with the broker PKI private key SK_B, set $U = D_{SK_B}(E_{PK_B}(rP)) = rP$.
- Step 2: Let $C_1 = < U, V >$ be a ciphertext encrypted using the IBE public key ID. Decrypt C_1 using the private key d: $V \oplus H(\hat{e}(d_{ID}, U)) = M$.

As mentioned above, TLES effectively ensures the secure transmission of secret keys, and only the proxy that matches the identity ID can get the keys. With time parameter *t*, it can effectively update the IBE private key of the proxy to improve security. Even if the control node has the IBE private key *d* of a proxy, it cannot recover rP because it does not have the PKE private key SK_B of the proxy. To get M, $H(g_{ID}^r)$ must be

obtained, since r is random, the control node cannot achieve its goal. If a malicious proxy replaces the public key PK_B, thus it has the corresponding PKE private key, which can get rP, but it has not the IBE corresponding private key d, $H(\hat{e}(d_{ID}, U))$ cannot be calculated, thus it cannot restore M .The security definitions of TLES scheme is shown in Appendix A. Our basic TLES scheme mentioned above is a one-way encryption secure scheme. According to [17], we can use a scheme from Fujisaki-Okamoto [19] to convert it to be secure against adaptive chosen-ciphertext attack conveniently.

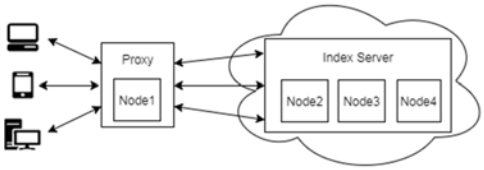
VI. PERFORMANCE EVALUATION

In this section we focus on three main questions. First, how much developer effort is required to construct our SSE schemes. Second, can our SSE schemes mitigate the inference attacks with the complete knowledge of the documents. Third, what are the performance overheads of IDCrypt on search operations and key sharing between different proxies.

A. Developer effort

For index construction, search and sharing scenarios, commonly used open source search engines such as Elastic Search [10] can be integrated into our system. We can manage the complexity of distributed systems well with the help of Elastic Search.

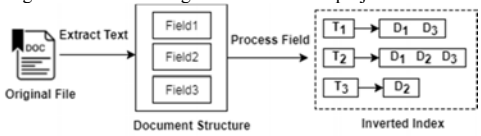
As shown in Fig 6, there exist multiple nodes in the IDCrypt system. IDCrypt builds the index in a node of a proxy, and balances the index data across the nodes of an index server to spread the index data and search load. The index server performs the search operation and returns the search results. It routes a search request from any proxy to the nodes that hold the index data, and returns the aggregated results to the proxy.



- Fig. 6 IDCrypt system

Elastic Search provides a distributed system on top of Lucene [11]. We also build the index of a document set in a proxy with the help of Lucene. Fig. 7 shows a traditional process of index building, in which an original file is first processed to a document structure that contains multiple fields. The fields are parsed to create keyword-docID pairs. When the size of keyword-docID pairs is sufficient large, the keyword–docID pairs with the same keyword are collected respectively to build a postings list of the keyword, where a posting is simply a docID. A docID is an integer which can be used to find the corresponding document contents. However, in our SSE scheme, as the data is encrypted and stored in the cloud, we need to map the docID to the document identifier that points to the encrypted data in the cloud. We do not need to manage the content of the original file, such that we can build the index easily and flexibly. As mentioned earlier, we generate the corresponding tokens for extracted keywords and encrypt the docID to a random value when processing each document, and store the searchable tokens and encrypted docIDs in the index.

In order to save memory or disk space, these docIDs that we just discussed are best stored in integer form, such that they can be compressed efficiently in the index. For instance, the posting lists of an index containing the integral docIDs can be compressed with delta-encoding [15]. To save time and space, in our SSE schemes, when a document is processed, its document identifier can be mapped to an integer range according to the number of keywords in the document. When extracting a keyword from the document, we make the docID plus one instead of encrypting the docID, and then build the token-docID pair. In this way we can construct an index of a document set securely and efficiently, and compress the index using the off-the-shelf algorithms in Lucene project.



- Fig. 7 Index building process

In the scenario of key sharing, the control node is deployed on a separate virtual machine. The communications between a broker/proxy and a control node is mainly implemented by socket transmission. We generate the PKI key pair with OpenSSL library [20]. We use elliptic curves to implement IBE, and invoke the PBC library [21] for bilinear pairings.

B. Security evaluation on our SSE schemes

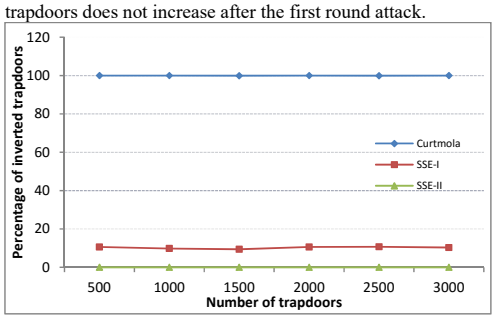
We empirically investigate the security of our SSE schemes by comparing them with Curtmola's construction. We use the count attack scheme [9] to infer the keywords of opaque query trapdoors, in which the adversary has the complete knowledge of all the documents. While it is unrealistic for an adversary to know all documents of a user in normal cases, this may happen sometimes. For example, a user has a set of emails stored at an email server, and it decides to encrypt all the emails using a SE scheme.

We use the online-available Enron [22] email set as the dataset. We chose the emails of the "_sent_mail" folder of 78 employees, resulting in 30,109 messages. A message is considered as a document. We extract the keywords in each document using the standard Porter stemming algorithm [23]; then remove stop-words [24] and duplicate keywords. There exist 49,982 unique keywords in the 30,109 documents. We then establish a fixed-size keyword universe from the keywords by taking the most frequent 5000 keywords. Fig. 8 shows the query recovery results of the count attack against Curtmola's scheme and our schemes. We run attacks on each scheme in the same setup with the adversary having no access

to any queries. For the fixed number of keyword universe, we vary the number of query keywords from 500 to 3000.

As the results show, an adversary can invert the most query trapdoors against Curtmola's construction, because it exposed search and access patterns. In this scheme, an adversary can deduce the keyword occurrence frequency and the keyword co-occurrence frequency over the keywords that have been queried. According to the count attack algorithm, for each search trapdoor, the adversary first counts the number of documents in its query result, and then tries to find a unique keyword appeared in the same number of plaintext documents. If the keyword is found, then it can be mapped to the trapdoor directly. We called this the *first round attack*. In the *second round attack*, the adversary uses the keyword co-occurrence frequency to deduce other mappings between keywords and trapdoors. Based on the already-built mappings, given an unknown trapdoor q, the adversary first selects the candidate keywords that appeared in the same number of plaintext documents as the length of query result of q. Then, to filter the candidate keyword set of q, for each pair of mapped keyword-trapdoor pair w' and q', the adversary computes the count c_1 of documents in which w and w' both appear in plaintext documents, and the count c_2 of documents which both the query q and q' hit in query results. If c_1 is not equal to c_2, then the keyword w will be removed from the candidate keyword set. Finally, if only one keyword meeting all the conditions is left, then it can be mapped to q.

In our SSE-I scheme, an adversary can invert a quite small portion of queried trapdoors, as the scheme only leaks the keyword occurrence frequency Information in the query results. The adversary can only perform the first round attack with such leakage. In particular, our SSE-II scheme leaks no information about the keyword occurrence frequency or the keyword co-occurrence frequency, thus the adversary can invert no search trapdoors. From Fig. 8 we can conclude that, when the size of the keyword vocabulary is fixed, even if only a few query trapdoors are inverted in the first round attack in Curtmola's scheme, then almost all trapdoors can be mapped to their plaintext keywords in the second round attack. However, in our SSE schemes, the number of identified trapdoors does not increase after the first round attack.



- Fig. 8 Query recovery results in different constructions

C. Performance overheads of IDCrypt

In this section we test the overheads of search and key sharing operations. The configurations of our test virtual machines are Intel 2.5 GHz dual-core with 4 GB of memory, the uplink speed is approximately 1 MB/s, and the downlink speed is approximately 7.5 MB/s. We use a macro-benchmarking framework Rally [25] to benchmark the search efficiency of our SSE scheme. Table I shows that, as the size and number of files increase, the index storage overhead grows slowly, and the search performance decreases lightly. Even for large file collections, for example, a size of 2.2GB, the index consumes just about 1.7GB storage space, and the search performance is about 11 operations per second.

TABLE I. SEARCH EFFICIENCY

Document Count	*Document Size (GB)*	*Index Size (GB)*	*Query (ops/s)*
9697882	1.212332	0.947194	19.5675
10716760	1.339540	1.05098	16.1885
11961342	1.487816	1.17934	14.6774
13053463	1.624238	1.25526	13.8309
17647279	2.201861	1.70634	10.9576

Finally, in order to demonstrate the performance of TLES scheme between different proxies, we test each process of the TLES, including system parameter generation, key issuing, key encryption, and key decryption. The average time in each process is shown in Table II. We concluded from the results that the decryption process including the PKE decryption process and the IBE decryption process takes more time than the encryption process, but it is negligible and not noticeable by a user. Generally speaking, the extra overhead in milliseconds level introduced by the IDCrypt is insignificant and acceptable for smooth user experience.

TABLE II. *TIME COST OF EACH PROCESS IN TLES*

	Params Gen	*Issuing*	*Enc*	*Dec*
Time (ms)	55	22	14	38

VII. RELATED WORK

A. Typical Searchable Encryption Schemes

Song et al. [4] proposed the first practical SE scheme. The search operation is simple, but the disadvantages are that the cloud requires full-text scanning and the computation is proportional to the data size. In addition, the server may use statistics to obtain additional user information.

Goh [5] proposed a secure index method to achieve SE using Bloom Filters. However, the Bloom Filter used in this scheme has an error rate, which may lead to inaccurate search results. It is not a sub-linear scheme.

Curtmola et al. [6] builds an encrypted search mechanism using inverted indexes, increased the search efficiency greatly, and improved the security of the encrypted search. It can only support exact keyword search and documents can not be updated dynamically. Based on this scheme, advanced SE functions are further developed, such as [12][26][27].

For multi-user encrypted search, Boneh et al. [28] realized the Public Key Encryption with Keyword Search (PEKS) algorithm using asymmetric encryption. This scheme led to greater performance loss.

However, all searchable encryption schemes mentioned in the above requires to modify the current cloud API. In addition, they leak search and access patterns for efficiency, which can be exploited by adversaries to get sensitive information [8][9].

B. Usable Searchable Encryption Systems

At protect-point *A* between a user and the client side in Fig. 2, ShadowCrypt [29] runs in the browser plug-in mode to perform encryption and search functions. ShadowCrypt only supports text input data, and do not support mobile platform. M-Aegis [30] proposes 7.5 layers between users and applications based on the mobile platform. M-Aegis only supports textual data. The SE functions of [29][30] are based on token-based SE scheme which exposed token occurrence patterns. An adversary can launch inference attacks on their SE schemes with prior knowledge using token occurrence patterns, as in [9][14].

At protect-point *B* between the client side and the server side in Fig. 2, Mylar [31] protects data from malicious server administrators based on the Meteor JavaScript framework, affecting the backward compatibility. Mylar used Song's SE scheme [4] to construct their SE functions, which can be attacked by recent works [32-33] using leaked information. In [32] an adversary can collude with some users to invert a query of an honest user, and [33] leverages design or implementation issues of [31] to infer sensitive information.

At protect-point *C* between the server side and the database in Fig. 2, CryptDB [34] encrypts the data before it is put into the database, and performs query requests on the encrypted data based on Song's SE scheme [4], thus effectively preventing the malicious database administrator. However, CryptDB can not prevent the server program behaving malicicouly.

C. Encrypted Key Sharing Schemes

If the data is encrypted with different keys, ciphertext sharing requires corresponding key sharing. Commonly used mechanisms for sharing keys are PKE and IBE. Two of the most well-known PKE certificate verification schemes are Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP). However, they are both inefficient in the case of revoking lots of certificates in real-time [18]. Micali proposed "Novomodo" system [35] achieving better efficiency, but it brings many third-party queries for certification status. To eliminate certificate status queries, IBE seems to be an effective way. Shamir [36] proposed the identity-based encryption and introduced an identity-based signature scheme. However, it is not practical for high-volume systems. Boneh [17] proposed a fully functional scheme based on the Weil pairing. Nevertheless, IBE itself has a private key escrow problem: The private key generator can decrypt the ciphertext of a user. To address this, Certificate-Based Encryption [18] and Certificate-Less Public Key Cryptography [37] combine IBE and PKE to realize double encryption, but they do not perform detailed performance evaluations. Lewko et al. [38] designed a scheme supporting multiple authorized parties. However, it has no practical evaluations. For sharing keys securely, we design a practical TLES scheme and perform performance evaluations.

VIII. Conclusion And Discussion

In this paper, we first conducted a systematic and quantitative comparison between index-based SE and token-based SE schemes. We describe the SSE schemes of our IDCrypt architecture in detail and analyze its security. The experiments results further show that IDCrypt indeed introduces fairly low overhead. To fulfill encrypted search, IDCrypt builds search indexes at proxies with the identifiers of encrypted data. We also design token-adjustment search schemes to search across different indexes. To share encrypted data between different proxies, we propose the two layers encryption scheme TLES to transmit secret keys. Certainly, IDCrypt still faces some technical challenges, and further research and improvement are needed. For example, due to the wide variety of cloud applications, we need to automatically match more protocols and build the index for them to search.

References

[1] C. S. Alliance, "CSA's Cloud Computing Top Threats in 2016" Cloud Security Alliance, Top Threats Working Group, February 2016. [Online]. Available:https://downloads.cloudsecurityalliance.org/assets/research/top-threats/Treacherous-12_Cloud-Computing_Top-Threats.pdf.

[2] http://breachlevelindex.com/assets/Breach-Level-Index-Report-H12016.pdf.

[3] Bösch C, Hartel P, Jonker W, et al. A survey of provably secure searchable encryption[J]. ACM Computing Surveys (CSUR), 2015, 47(2): 18.

[4] Song D X, Wagner D, Perrig A. Practical techniques for searches on encrypted data[C]//Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. IEEE, 2000: 44-55.

[5] Goh E J. Secure Indexes. International Association for Cryptologic Research Cryptology ePrint Archive, 2003: 216.

[6] Curtmola R, Garay J, Kamara S, et al. Searchable symmetric encryption: Improved definitions and efficient constructions. Journal of Computer Security, 2011, 19(5):79-88.

[7] Goldreich O, Ostrovsky R. Software protection and simulation on oblivious RAMs[J]. Journal of the ACM (JACM), 1996, 43(3): 431-473.

[8] Islam M S, Kuzu M, Kantarcioglu M. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation[C]//NDSS. 2012, 20: 12.

[9] Cash D, Grubbs P, Perry J, et al. Leakage-abuse attacks against searchable encryption[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015: 668-679.

[10] Elastic Search. https://www.elastic.co/products/elasticsearch.

[11] Lucene. https://lucene.apache.org/

[12] Kamara S, Papamanthou C, Roeder T. Dynamic searchable symmetric encryption[C]//Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012: 965-976.

[13] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–679, May 1983.

[14] Pouliot D, Wright C V. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016: 1341-1352.

[15] Schütze H. Introduction to Information Retrieval[C]//Proceedings of the international communication of association for computing machinery conference. 2008.

[16] Atallah M J, Blanton M, Fazio N, et al. Dynamic and efficient key management for access hierarchies[J]. ACM Transactions on Information and System Security (TISSEC), 2009, 12(3): 18.

[17] Boneh D, Franklin M. Identity-based encryption from the Weil pairing[C]//Annual International Cryptology Conference. Springer Berlin Heidelberg, 2001: 213-229.

[18] Gentry C. Certificate-based encryption and the certificate revocation problem[C]//International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 2003: 272-293.

[19] Fujisaki E, Okamoto T. Secure integration of asymmetric and symmetric encryption schemes[C]//Crypto. 1999, 99(32): 537-554.

[20] OpenSSL. https://www.openssl.org/.

[21] PBC: Pairing Based Cryptography. http://crypto.stanford.edu/pbc/.

[22] Enron email dataset. https://www.cs.cmu.edu/~./enron/. Accessed: 2015-05-13.

[23] M. Porter. An algorithm for suffix striping. Program, 14(3):130–137, 1980.

[24] Common-English-Words. http://www.textfixer.com/tutorials/common-english-words.txt/.

[25] Rally. https://github.com/elastic/rally.

[26] Xia Z, Wang X, Sun X, et al. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(2): 340-352.

[27] Li J, Wang Q, Wang C, et al. Fuzzy keyword search over encrypted data in cloud computing[C]//INFOCOM, 2010 Proceedings IEEE. IEEE, 2010: 1-5.

[28] Boneh D, Di Crescenzo G, Ostrovsky R, et al. Public key encryption with keyword search//Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques.Berlin Heidelberg,Germany, 2004: 506-522.

[29] He W, Akhawe D, Jain S, et al. Shadowcrypt: Encrypted web applications for everyone[C]//Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014: 1028-1039.

[30] Lau B, Chung S, Song C, et al. Mimesis aegis: A mimicry privacy shield–a system's approach to data privacy on public cloud[C]//23rd USENIX Security Symposium (USENIX Security 14). 2014: 33-48.

[31] Popa R A, Stark E, Valdez S, et al. Building web applications on top of encrypted data using Mylar[C]//11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). 2014: 157-172.

[32] Van Rompay C, Molva R, Önen M. A Leakage-Abuse Attack Against Multi-User Searchable Encryption[J]. Proceedings on Privacy Enhancing Technologies, 2017, 3: 164-174.

[33] Grubbs P, McPherson R, Naveed M, et al. Breaking web applications built on top of encrypted data[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016: 1353-1364.

[34] Popa R A, Redfield C, Zeldovich N, et al. CryptDB: protecting confidentiality with encrypted query processing[C]//Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011: 85-100.

[35] Micali S. NOVOMODO: Scalable Certificate Validation and Simplified PKI Management[J]. Annual Pki Research Workshop, 2002.

[36] Shamir A. Identity-based cryptosystems and signature schemes[C]//Workshop on the Theory and Application of Cryptographic Techniques. Springer Berlin Heidelberg, 1984: 47-53.

[37] Al-Riyami S S, Paterson K G. Certificateless public key cryptography[C]//International Conference on the Theory and Application of Cryptology and Information Security. Springer Berlin Heidelberg, 2003: 452-473.

[38] Lewko A, Waters B. Decentralizing attribute-based encryption// Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques. Tallinn, Estonia, 2011: 568-588.

APPENDIX

A. SECURITY OF ONE-WAY ENCRYPTION TLES SCHEME

To give the security notion of One-Way Encryption (OWE) TLES scheme, we first give the security notions of OWE Public-Key Encryption (PKE) [19] and OWE Identity-Based Encryption (IBE) [17].

Security of OWE PKE: Let Π = *(K, E, D)* be a PKE scheme. Let *A* be a probabilistic polynomial-time adversary. Given a sufficiently large *s*, we say that Π is OWE secure if for every PPT adversary *A*, all polynomials *p*, the advantage of *A* is

$$\mathrm{Adv}^{\mathrm{owe}}_{A,\Pi}(k) \triangleq \Pr\left[(pk, sk) \leftarrow \mathcal{K}(1^k); x \leftarrow \mathrm{MSP}; y \leftarrow \mathcal{E}_{pk}(x) : A(pk, y) = \mathcal{D}_{sk}(y)\right],$$

< 1/p(s)

In which MSP represents message spaces determined by *s*. The message *x* is picked up uniformly from MSP to compute a ciphertext *y*. Then the adversary *A* outputs a string on input (*pk*, *y*) as the decryption of *y* with *pk*.

Security of OWE IBE: An identity-based encryption scheme is ID-OWE secure if no PPT adversary A has a non-negligible advantage in the following game:

Setup: The challenger takes a security parameter *s* as input and runs the Setup algorithm. It gives the resulting system parameters *params* to the adversary, and keeps the master-key private.

Phase 1: The adversary issues private key extraction queries adaptively for $ID_1, \ldots, ID_m$. The challenger runs *Extract* algorithm to generate the private key d_i corresponding to ID_i and sends d_i to the adversary.

Challenge: When Phase 1 is over, the adversary outputs a public key $ID \neq ID_1, \ldots, ID_m$ on which to be challenged. The challenger picks a random $M \in M$ and encrypts it using *ID* as the public key. Then the resulting ciphertext *C* is sent to the adversary.

Phase 2: The adversary issues more extraction queries $ID_{m+1}, \ldots, ID_n$ adaptively with the constraint that $ID_i \neq ID$. The challenger responds as in Phase 1.

Finally, the adversary outputs a message $M' \in M$. We say that Π is ID-OWE secure if for every PPT adversary *A*, all polynomials *p* and sufficiently large *s*, the advantage of *A* is

$$Adv(A) = Pr[M = M'] < 1/p(s)$$

We are concerned about two different types of attacks referring to [18]: 1) by an uncertified client and 2) by the certifier, and we define two different games accordingly. In Game 1, the adversary has PKE public-private key pair and its IBE public key (including identity *ID* and possibly time parameter *t*), but does not know the IBE private key SK_{IBE} corresponding to the public key *params*. It can make extraction and decryption queries. In Game 2, the adversary acts as the role of the certifier (PKG). It has the PKE public key *PK*, IBE public key *params* and its private key SK_{IBE}, but does not have the PKE private key *SK*. It can make decryption queries. We say that our TLES scheme is OWE secure if no adversary can win either game.

Game 1: The challenger takes a security parameter s and runs the Setup algorithm of IBE. It gives the adversary the resulting system parameters *params*. It keeps the master-key SK_{IBE} to itself. Then the adversary interleaves extraction and decryption queries with a single challenge query. These queries are answered as follows:

--On extraction query $<s,ID_i,PK,SK>$, the challenger runs *Extract* algorithm to generate the private key d_i corresponding to the public key ID_i. It sends d_i to the adversary.

--On decryption query $<s,ID,PK,SK,C>$. The challenger responds by running algorithm *Extract* to generate the private key d_i corresponding to ID_i. It then runs algorithm *Decrypt* to decrypt the ciphertext C using the private key d_i and SK. It sends the resulting plaintext to the adversary.

-- On challenge query $<s',ID',PK',SK'>$, the challenger picks a random $M \in M$ and runs *Encrypt* algorithm to encrypt it using ID' and PK'. Then the resulting ciphertext C' is sent to the adversary.

Finally, the adversary outputs a message $M' \in M$. We say that the TLES is OWE secure against Game1 if for every PPT adversary A, all polynomials p and sufficiently large s, the advantage of A is

$$Adv(A) = Pr[M = M'] < 1/p(s)$$

Proof: Let A be a PPT adversary that has advantage $\epsilon(s)$ against TLES. Suppose A makes at most q_E private key extraction queries and at most q_D decryption queries. Then there is a PPT adversary B that has advantage at least $\epsilon(s)$ against OWE IBE. Its running time is $O(time(A))$.

We construct an adversary B that uses A to gain advantage $\epsilon(s)$ against OWE IBE. Adversary B interacts with A as follows:

Phase 1: At any time adversary A issues extraction queries, adversary B receives the queries and sends them to the IBE challenger. The challenger runs algorithm *Extract* to get the private key d_i corresponding to the public key ID_i. It sends d_i to the adversary B, and then B sends it to A.

At any time adversary A issues decryption queries, the adversary B decrypts the ciphertext of *'rP'* using the private key SK to get *'rP'*, then sends remaining data and *'rP'* to the IBE challenger. The challenger responds by running algorithm *Extract* to generate the private key d_i corresponding to ID_i. It then decrypts the remaining data using the private key d_i. It sends the resulting plaintext to adversary B, and then B sends it to A.

Once the adversary A decides that Phase 1 is over it outputs an identity ID' on which it wishes to be challenged. The only constraint is that ID' did not appear in any private key extraction queries and decryption queries in Phase 1.

The adversary B sends ID' to IBE challenge. The IBE challenger picks a random $M \in M$ and encrypts M using ID'. It sends C' as the challenge to the adversary B, Then B encrypts *'rP'* of C' using PK' and sends it to A.

Phase 2: The adversary issues more queries where query q_i is one of:

--extraction query in which ID_i != ID'. The challenger responds as in Phase 1.

--decryption query in which ID_i != ID' and C_i != C'. The challenger responds as in Phase 1. These queries may be asked adaptively as in Phase 1.

Finally, the adversary A outputs a guess M' and wins the game if $M = M'$. The adversary B outputs the same M' and wins the game if A wins. So, if A is an adversary that has advantage $\epsilon(s)$ against TLES, then B is an adversary that has advantage $\epsilon(s)$ against OWE IBE.

Game 2: The challenger takes a security parameter s and runs the Setup algorithm of PKE. It gives the adversary the resulting PK. It keeps the SK to itself. Then the adversary interleaves decryption queries with a single challenge query. These queries are answered as follows:

--On decryption query $<s,ID_i,params,SK_{IBE},C_i>$, the challenger responds by running algorithm *Extract* to generate the private key d_i corresponding to ID_i. It than runs *Decrypt* algorithm to decrypt the ciphertext C_i using the private key SK and d_i. It sends the resulting plaintext to the adversary.

--On challenge query $<s',ID',params',SK_{IBE}'>$, the challenger picks a random $M \in M$ and runs *Encrypt* algorithm to encrypt it using ID' and PK. Then the resulting ciphertext C' is sent to the adversary.

Finally, the adversary receives C' and outputs a message $M' \in M$. We say that the TLES is OWE secure against Game2 if for every PPT adversary A, all polynomials p and sufficiently large s, the advantage of A is

$$Adv(A) = Pr[M = M'] < 1/p(s)$$

Proof: Let A be an adversary that has advantage $\epsilon(s)$ against TLES. Suppose A makes at most q_D decryption queries. Then there is an adversary B that has advantage at least $\epsilon(s)$ against OWE PKE. Its running time is $O(time(A))$.

We construct an adversary B that uses A to gain advantage $\epsilon(s)$ against OWE PKE. Adversary B interacts with A as follows:

Phase 1: The adversary issues decryption queries where query q_i is:

At any time adversary A issues decryption queries, the adversary B runs algorithm *Extract* to generate the private key d_i corresponding to the public key ID_i. Then B sends ciphertext of *'rP'* to the PKE challenge. The challenge decrypts the ciphertext of *'rP'* using the private key SK. Then the PKE challenge sends *'rP'* to B. B then decrypts the remaining data using the private key d_i and *'rP'*. It sends the resulting plaintext to the adversary A.

Once the adversary A decides that Phase 1 is over, it issues a challenge query $<s',ID',params',SK_{IBE}'>$ to the adversary B. The adversary B sends the challenge to PKE challenger. The challenger picks a random $M \in M$ and encrypts M using PK and ID' then sends C' to the adversary B, Then B sends C' to A.

Phase 2: The adversary issues more decryption queries where C_i != C'. The PKE challenger responds as in Phase 1. These queries may be asked adaptively as in Phase 1.

Finally, the adversary A outputs a guess M' and wins the game if $M = M'$. The adversary B outputs the same M' and

wins the game if A wins. In Game2 the adversary acts as the role of the certifier (PKG). It has the IBE public key and its private key, but does not have the PKE private key. So, if A is an adversary that has advantage $\epsilon(s)$ against TLES, then adversary B has advantage $\epsilon(s)$ in decrypting the ciphertext of *'rP'* in *C'* without *SK*, thus B is an adversary that has advantage ϵ(s) against OWE PKE.

B. COUNT ATTACK ALGORITHM

The count attack scheme is shown in Algorithm 3. In line1, for each search trapdoor, the adversary first counts the number of documents in its query result, and then seeks to find a unique keyword appeared in the same number of plaintext documents. If the keyword is found, then it can be mapped to the trapdoor directly. In line 2, the algorithm builds a query co-occurrence counts C_q, where $C_q[i, j]$ represents the number of documents query q_i and q_j both match. Similarly, a keyword co-occurrence counts C_I can also be built, where $C_I[i, j]$ represents the number of plaintext documents in which keyword w_i and keyword w_j both appear.

Algorithm 3: The Count Attack Algorithm

Input: query trapdoors T and results, unencrypted keyword index I;
Output: mapping set between keywords in I and trapdoors in T;

```
1   initialize the base mapping set K;
2   compute query co-occurrence matrix C_q for trapdoors T and
    keyword co-occurrence matrix C_I for index I;
3   while size of K is increasing do
4       for each unknown trapdoor t ∈ T-K do
5           build candidate keyword set S = {s : the occurrence count of
            s equals to the occurrence count of t };
6           for s ∈ S do
7               for known base mapping (t', s') ∈ K do
8                   if C_q[t, t'] ≠ C_I[s, s'] then
9                       remove s from S;
10          if one keyword s remains in S then
11              add (t, s) to K;
12  return the mapping set K;
```