Secure Multi-Client Data Access with Boolean Queries in Distributed Key-Value Stores

Xu Yuan[†] Xingliang Yuan[‡] Baochun Li[†] Cong Wang[‡]
[†] Department of Electrical and Computer Engineering, University of Toronto
[‡] Department of Computer Science, City University of Hong Kong, China

***Abstract*—In the era of big data processing, it is desirable to manage large volumes of data with high scalability, confidentiality protection, and flexible types of search queries. In this paper, we propose a design to store encrypted data on a cluster of distributed servers while supporting secure and authorized Boolean queries. In particular, the data owner encrypts the database with encrypted searchable index attributes, and the encrypted data values are stored evenly across multiple servers by leveraging a distributed index framework. Based on this design, we show how to construct encrypted indexes, generate search tokens, and query parallelly to achieve efficient Boolean search. Moreover, these queries are not only limited to those initiated by the data owner but also by other authorized clients. Specifically, we further integrate a recent scheme to make the authorization of client's requests non-interactive. The data owner is not required to stay online to interact with the clients. We characterize the leakage profile and provide a formal security analysis to demonstrate that our system can guarantee data confidentiality and query privacy. To validate our protocol, we implement a system prototype and evaluate the efficiency of our construction experimentally. Through experimental results, we show the effectiveness of our protocol in term of data encryption time and Boolean query time.**

I. Introduction

The last decade has witnessed a significant increase in the volume of data generated every day, reaching an order of exabytes. Applications of big data have been extended into many areas, such as social networking, e-health systems, and smart grid systems. It is becoming increasingly common for data to be hosted off-site, especially with the rise of cloud computing. However, the need for storing such large volumes of data in the cloud has raised new challenges for its scalability, privacy, and support for flexible types of data operations.

To guarantee security and privacy, data should be encrypted before outsourcing to remote servers. This brings barriers for data owners or other clients to perform flexible data operations on the encrypted data. Particularly, even though standard encryption technology could protect data confidentiality, it does not readily support textual search functions over encrypted data. Many recent mechanisms, in the general category of Searchable Symmetric Encryption (SSE), have been introduced to enhance data privacy while preserving data operation privileges (e.g., [1]–[3]). These works focused on either centralized servers or simple data operations (e.g., single keyword search), and do not address all the new requirements on storage scalability, security and privacy guarantee, as well as fast data retrieval in big data applications.

The goal of this paper is to address the challenges imposed by big data applications on system scalability, data confidentiality, and support for flexible data operations. To improve storage scalability, we employ a distributed index framework (i.e., key-value (KV)) to encrypt and distribute data evenly across multiple servers. To keep data private, all data should be encrypted as long as they are left from the data owner. At the same time, the search operations as in the plaintext systems should still be preserved. Also, these operations are not only limited to those initiated by the data owner, and should also be extended to support multiple clients, where authorized clients can access the private database owned by a data owner. In particular, we study how to achieve secure and efficient Boolean search in a multi-client setting to meet the needs of real-world data applications.

Boolean search is intensively used in big data applications. For example, in hospital database systems, if the doctors require investigating all male patients with the diabetes clinical trial, they will perform the conjunctive search for keywords "male" and "diabetes". In the encrypted domain, Boolean search can be realized to search individual keywords. By doing so, prior SSE schemes for single keyword search [1]–[3] can directly be used. However, this treatment will introduce two issues. The first is that the search time scales in the number of keywords in a Boolean query. If some keywords match a large portion of documents in a dataset, it will incur a long query latency. The other is that the server learns all the matched documents of each keyword, even some of the documents are not in the final query results, and are not expected to be revealed. Although there are some recent efforts [4], [5] working on these issues, how to apply them to an encrypted and distributed data store remains an open challenge.

On the other hand, enabling authorized keyword search for multi-clients is also critical to data applications in practice. For example, in the hospital system, one hospital has the rich information about clinical trials "diabetes" and would like to share this information with partner hospitals. It is useful to authorize other hospitals to access its database of the diabetes information but without leaking additional information. One solution is to use broadcast encryption as introduced in [2], but it is not scalable for a large number of users, and supports single keyword search only. Recent solutions in [6], [7] support Boolean search in specific access policies, but they ask the client to communicate with the data owner for authorization of each query. This somehow limits the throughput of the entire system.

In this work, we focus on enabling authorized Boolean search in encrypted and distributed data stores. First, we start from the recent advancement of encrypted Boolean search [5] that enables conjunctive search in sublinear complexity. To deploy this primitive to a distributed data store, we leverage the framework of an encrypted KV store [8], so as to empower the parallel computing ability of distributed systems for the encrypted queries. In particular, our design allows the data owner to build the local encrypted databases (LEDB) and Boolean search encrypted index (LBSIndex) that will be stored on each server, where the LEDB on a server stores the encrypted documents while the corresponding LBSIndex indexes those documents for Boolean queries. As a result, a given Boolean query can be processed by all the servers in parallel.

In a multi-client setting, to control clients' access privileges of Boolean queries, we further integrate the latest multi-client SSE scheme [9] into our design, and advocate this non-interactive scheme, where a client only needs to interact with the data owner once to request the necessary search keys from the access policy, and later generate the permitted search tokens without the interaction with the data owner. In order to set up our design in a production system, we show how to implement the proposed protocols and cryptographic constructions in a known key-value store, i.e., Redis. Specifically, the LEDB and LBSIndex are physically stored as key-value pairs in Redis, and are accessed by native Redis API, i.e., `Put`, `Get`, and `Exists`.

Based on our design, we characterize the leakage profile incurred in the Boolean search, and show that the information leakage is no more than that from the current state-of-the-art works, i.e., [5], [9]. We provide a formal security analysis to show that data privacy is robust (i.e., leakage is no more than defined) to the potential attackers. To show the effectiveness and efficiency of our theoretical results, we have developed a system prototype and evaluated the performance of our construction experimentally. Through experimental results, we have validated the correctness and demonstrated the effectiveness of our scheme.

The main contributions of this paper can be summarized as follows:

- We design a secure encryption scheme to support Boolean search based on a distributed index framework. The local encrypted database and search structures are built and stored on multiple servers in a distributed fashion to be more scalable. Due to the characteristics of local encryption structures, we demonstrate that our protocol enables parallel search and computation for Boolean queries among all servers. Therefore, the search efficiency is significantly improved. On the other hand, the communication overhead and workload on each single server can be significantly reduced when comparing to a centralized single server due to the parallel operation, which eventually improve network throughput.
- We advocate the non-interactive multi-client access control scheme and integrate it into our design, where the client only needs to interact with a data owner once to obtain the necessary search keys. In this scheme, both data owner's data and clients' search information can be protected. With these keys, a client can always generate the token by itself to perform Boolean queries without interacting with data owner.
- We formally show the security and privacy guarantee of the newly designed protocol. By characterizing and analyzing the leakage profile from both the encryption structure and query process, we show that the information leakage is no more than that from the current state-of-the-art works. We also discuss the existing attack models and address that our protocol is robust to these models.
- We design a system prototype and deploy it to the Amazon EC2 to validate our design. The experimental results show that both the data uploading time and Boolean query efficiency are significantly improved when comparing to a centralized server and the interactive scheme.

The remainder of this paper is organized as follows. In Section II, we introduce the system model for this paper. Section III discuss the main challenges of Boolean search and multi-client query over distributed KV store. Section IV proposes our construction of secure KV store with the supports of multi-client access and Boolean search. Specifically, we present conjunctive search as an example to show how the encrypted local index is built, search tokens generated, and parallel queries performed. In Section V, we characterize the leakage profiles and discuss the maximum leakage incurred to adversaries and clients in the encryption scheme and query process. In Section VI, we present the implementation and performance evaluation of our protocol. Section VII discusses related work and Section VIII concludes this paper.

II. System Model

A. Overview

In this section, we describe a system architecture that supports secure multi-client data access and Boolean search over distributed and encrypted KV store. As shown in Figure 1, the system model consists of four entities: data owner, clients, dispatcher, and a number of distributed public cloud servers. The former two entities are individuals or enterprises users, and the following two entities are public cloud service nodes. The data owner is a user who wishes to store its encrypted data into the public cloud service nodes. The rule of a dispatcher is to route the encrypted documents evenly across all distributed servers. In our architecture, there are multiple clients who wish to access the data owner's database but have to request the authorization from the data owner.

In our setting, we assume the data owner has a private database which includes the documents, document IDs, and keywords. This database is required to store on remote servers

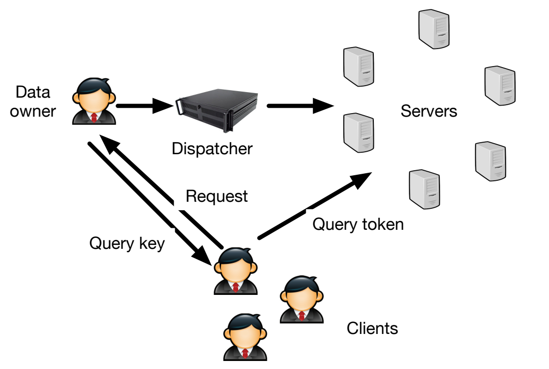

Fig. 1: The system architecture.

and the confidentiality should be guaranteed. For each document, it has a unique ID and a set of keywords $\mathbf{w}$. To preserve the confidentiality and privacy, the data owner encrypts its data and build the searchable index to enable the query operation. Since the database will be distributively stored on multiple servers, the data owner needs to build the local encryption database and local searchable index respected to each single server, and sends them to the dispatcher to decide which server will be routed to. The data owner can perform the queries either with document IDs or keywords to find the matched documents. In our model, we assume these queries should not only be limited to data owner but also can be extended to multiple clients. That is, the data owner can authorize other clients to access its database.

In this work, we focus our study on how to enable client to perform the Boolean search, i.e., conjunctive of multiple keywords. For example, we wish to find the conjunctive results of keywords $w_1, w_2, \cdots, w_m$., i.e., $w_1 \wedge w_2 \wedge \cdots \wedge w_m$. If one client's search request is authorized, it can submit the conjunctive search request $w_1 \wedge w_2 \wedge \cdots \wedge w_m$ to servers. Upon receiving this request, all servers perform the queries and return all matched documents that include all keywords $w_1, w_2, \cdots, w_m$.

B. Threat Model

In this paper, we aim to protect data owner's database. Regarding the privacy and confidentiality, we are targeting the threats from semi-honest adversaries (both servers and clients). The data owner will never expose its encryption keys to the servers and other unauthorized clients. Besides, we assume our distributed KV store will not allow the attackers to access data owner's private keys. Our designed scheme is secure against the passive attacker, who can fetch the encrypted database but cannot obtain more information rather than encrypted indexes. For the positive attackers, we assume they can monitor the query protocol and analyze both the query pattern and result pattern. In this paper, we do not consider the case where attackers can access the background information about the queries and database. Thus, inference attack [10] and leakage-abuse attack [11] will not be included here. Also, we do not consider the case where malicious attackers can modify or delete the database intentionally.

C. Background Knowledge

Symmetric Encryption: A symmetric encryption scheme $(KGen, Enc, Dec)$ contains three algorithms: The key generation algorithm $KGen$ takes a security parameter λ to return a secret key K. The encryption algorithm Enc takes a key K and a message $m \in \{0,1\}^*$ to return a ciphertext $m^* \in \{0,1\}^*$; The decryption algorithm Dec takes K and m^* to return m.

Pseudo-random Function: We define a family of pseudo-random functions $F : \mathcal{K} \times X \rightarrow R$, if for all probabilistic polynomial-time distinguishers D, $|Pr[D^{F(k,\cdot)} = 1 | k \leftarrow \mathcal{K}] - Pr[D^g = 1 | g \leftarrow \{\mathsf{Func} : X \rightarrow R\}]| < negl(k)$, where $negl(k)$ is a negligible function in k.

Encrypted key-value store: We follow the construction of encrypted key-value stores proposed in [8], where the encrypted document can be stored as a key-value pair. Assume that the data owner has a set of documents $\mathbf{f}$, and for each document f_i, it has a unique document identifier id_i. The documents identifiers $\mathbf{id}$ are protected with PRF P, and the documents $\mathbf{f}$ are encrypted with the symmetric encryption algorithm. Then the entry of KV store is defined as:

$$< l^*, f^* > = < P(K_{id}, id), Enc(K_f, f) > \qquad (1)$$

where K_{id} and K_f are the private keys. The dispatcher can use consistent hashing to find the target server for l^*. With consistent hashing, all documents can be stored across multiple nodes with a balanced distribution.

III. Boolean Search and Multi-client Query over Distributed KV Store

Before we present our construction, we discuss the straightforward methods to support multi-client query and Boolean search, which will raises the challenges should be addressed in the design. This section also serves as a demonstration of the necessity and importance of constructing a more secure scheme to support multi-client data operations.

A. Boolean Search

Based on our system model, one straightforward method to execute Boolean search is to reduce the conjunctive search into m single keyword search and return a set of matched documents for each keyword. Upon receiving all matched documents sets, the client performs the intersection operation to find the common documents among these sets. That is, the client performs the search for each keyword w_i and returns a set of matched encrypted documents (denoted as set F_i) to the clients. With all returned encrypted document sets F_i where $i = 1 \cdots m$, the clients can find the intersection among $F_1, F_2, \cdots, F_m$, i.e., $F_1 \cap F_2 \cdots \cap F_m$. This method is simple but inefficient since it returns too many redundancy documents. The complexity is the sum of all matched documents for all search keywords. Especially, if one keyword is included in all documents, the whole database will be returned. On the other hand, it will cause the severe network overhead. More importantly, this method incurs significant information leakage, i.e., the set of documents matching each keyword.

```
Conjunctive search
1. Input: Private key $K$, $DB$, $\{w_1 \wedge w_2 \wedge \cdots \wedge w_m\}$.
2. begin:
3.    $K_{w_i} \leftarrow H(K, w_i)$;
4.    For $i = 1 \cdots m$
5.       $t_1^i \leftarrow F_1(K_{w_i}, 1||w_i), t_2^i \leftarrow F_2(K_{w_i}, 2||w_i)$;
6.    send $\{t_1^i, t_2^i\}_{i=1}^n$ to cluster storage nodes
7.    for node z= $1, \cdots, n$ do
8.    while(true)
9.       $c_i \leftarrow 1$;
10.      $\alpha \leftarrow G_1(t_1^1, c_i)$
11.      while $(\alpha, \cdot) \in XSet_z$ do
12.         $\beta \leftarrow (\alpha, \cdot)$;
13.         $l^* \leftarrow \beta \oplus G_2(t_2^1, c_i)$;
14.         for $j = 1, \cdots, m$ do
15.            if(($G_1(t_1^j, c_i), G_2(t_2^j, c_i) \oplus l^*) \notin XSet_z$)
16.               break;
17.            $f^* \leftarrow get(l^*)$, $\mathcal{F} \leftarrow \mathcal{F} \cup f^*$
18.      end while
19.      if($(\alpha, \cdot) \notin XSet_z$)
20.      break;
21.         else
22.         $c_i++$;
23.      $\alpha \leftarrow G_1(t_1^1, c_i)$
24.      send $\mathcal{F}$ to client.
25.      end while
```

Fig. 2: Support boolean queries.

To overcome weakness above, one consideration is to let servers to perform the Boolean search and only return intersection results to the client. By observing the data structure of KV store in [8], we can see that α is corresponding to the keyword attributes and β is corresponding to the matched document id. Therefore, the server can check whether two keywords have the same index l^*, and thus find the intersection results between them. In one word, the client can submit all keywords query request to servers, and servers can perform the intersection operation for (α, β) sets to find the matched document indexes. Through this way, the servers can only return the matched conjunctive search results to the clients. The Detailed description of this scheme is shown in Figure 2.

Comparing to method above, this scheme is more efficient since only matched documents are returned to the client, which greatly reduces the network overhead. However, since the client submits all (t_1, t_2) to the server, the honest-but-curious servers can still learn the number of document matching each keyword, as well as the number of document matching the conjunctive query of any combination of a subset of keywords in $\{w_1, w_2, \cdots, w_m\}$.

B. Multi-client Query

To enable multi-client query, one method is to allow data owner to send all encryption keys to clients. This method is simple, but brings risk to the encrypted databases. With the encrypted key, clients can access the whole databases, or even modify the database. In this situation, the data owner will lose the control of its own database, which definitely should not happen.

Another secure method is that the clients submit query request to the data owner. Upon receiving the request, the data owner can generate the query token and sends back to the client. With query token, clients can perform the query operation with the obtained token from data owner to cloud servers. This method is simple but requiring data owner to stay online to process per-query and generate query token responding to client's request. On the other hand, the query information from the client will be learnt by the data owner.

From the discussion above, we conclude that, to support multi-client query and Boolean search, a more secure and efficient scheme is needed.

IV. The Proposed Construction

In this section, we present our design to support multi-client Boolean queries in a distributed and encrypted KV store to satisfy the efficiency and secure requirements. First, we introduce how to deploy an SSE scheme specifically designed for Boolean queries to an encrypted KV store with a distributed index framework. Second, we integrate a non-interactive scheme to enable authorized Boolean queries in the multi-client setting.

The idea of the scheme [5] is to build two indexes to achieve sublinear search time with minimized leakage for conjunctive queries. From the high-level point of view, the first is an encrypted index, where each entry is an encrypted keyword associated with a set of matched document IDs. The second index is a file-word index, where every matched id in each inverted list is linked to a set of keywords in this document. To implement this scheme in a KV store, we first use consistent hashing to determine the server location of each document, and then transform this encrypted invert index into an encrypted KV pairs, where the key is an encrypted keyword with a state, and the value is an encrypted document id. For the file-word index, it is also transformed into KV pairs, where the key is derived from the document and a keyword of this document while the value indicates the existence.

To enable authorized queries, we further adopt a secure non-interactive scheme proposed in [9]. In this scheme, a client is only required to interact with the data owner in the setup stage to obtain the necessary keys within a specified search policy. After that, the client can generate the search token to perform the permitted Boolean queries without the interaction with the data owner. Note here, with these keys, the clients cannot execute more data operations (e.g., modify database) rather than only performing query operations within the policy.

A. Our Construction

To enhance search efficiency, we consider the search respected to keyword and document ID pairs instead of keyword document pairs. Since the server stores both document ID and document pairs, when a document ID is located, the respected documents can be quickly retrieved. Thus, the search for a document ID is the same as for a document itself.

1) Build Encrypted Database: To improve memory efficiency, the data owner employs the hash functions to transform keywords into prime integers. That is, for each keyword, it has a unique corresponding prime integer. At the server side, it will store prime integers rather than keyword strings. Since this part is out of the scope of this paper, the detailed discussion of keywords to prime integers transformation is omitted here to conserve space. In this paper, we assume data owner already has the keywords and corresponding prime integers pairs $(\mathbf{w}, \mathbf{P})$, i.e., $(P_1, P_2, \cdots, P_S) \leftarrow (w_1, w_2, \cdots, w_S)$. In the following section, we will represent P as the keyword for easy of discussion.

In the distributed KV store, since the encrypted database is stored across multiple servers, it is important to build local index sets rather than the global index as in [5]. Particularly, we build the local encrypted database (LEDB) and Local Boolean search index (LBSIndex) corresponding to each server. Now, we show how to build local LEDB and LBSIndex. To generate necessary keys for encrypted document ID and keyword, data owner inputs a security parameter κ, Then, it chooses big primes p, q, random keys K_a for a hash function H, K_I, K_Z, K_X for a PRF F_p and K_P for a PRF F. With these keys, it outputs the system master key $MK = (p, q, K_w, K_I, K_Z, K_X, g_1, g_2, g_3, msk)$ and the corresponding system public key $PK = (n, g, mpk)$, where $(mpk, msk) \leftarrow ABE.Setup(1^\kappa)$, $n = pq, g \overset{\$}{\leftarrow} G$ and $g_i \overset{\$}{\leftarrow} Z_n^*$ for $i \in [3]$

For each server i, we initiate the LEDB$_i$ and LBSIndex$_i$ as the empty sets at each server i. For each document, it has a unique ID l^*, we store the key-value pair as l^* and P (P corresponding to a keyword) where l^* corresponding to the document ID. As we mentioned, when the document ID is located, the document can be quickly retrieved. Since we have used l^* as the search attribute, now we can store the pair between P and each respected l^*. The relationships between l^* and keyword P (i.e., prime integer) are the main objectives that we need to protect. Since each P will be contained in multiple documents, we increment a value c from 1 with each respected l^*. For each l^*, dispatcher will decide the target server i which will be routed to for the encrypted keyword and l^*, we have $i \leftarrow route(l^*)$. To protect l^* and keyword relationships, we build local LEDB on each server i in the following format: $e \leftarrow Enc(mpk, l^* || k_{id}, S)$, xind $\leftarrow F_p(K_I, l^*)$; and $z \leftarrow F_p(K_Z, g_2^{1/P} \mod n || c_j)$, where k_{id} is the document encryption key. We then have $y \leftarrow \text{xind} \cdot z^{-1}$. To store them into LEDB, we set $stag_P \leftarrow F(K_P, g_1^{\frac{1}{P}} \mod n)$ and $label \leftarrow F(stag_P, c_j)$. Then, we add (e, y) to LEDB$_i$, i.e., LEDB$_i[label] \leftarrow (e, y)$. To build LBSIndex, we set xtag $\leftarrow g^{F_p(K_X, g_3^{1/P} \mod n) \cdot \text{xind}}$ and add xtag into LBSIndex, i.e., LBSIndex$_i \leftarrow$ LBSIndex$_i \cup \{\text{xtag}\}$ respected to server i. This progress continues till all keywords and their corresponding document IDs are added into LEDB and LBSIndex. Fig. 3 shows the details of building local LEDB and LBSIndex. Now, we have encrypted database LEDB$_i$ and LBSIndex$_i$ corresponding to each server i, and the dispatcher will route them to the corresponding servers and locally store them.

```
Secure Put
1. Request: Put(K, f)
2. Data: Private key $K$, $DB$, $\mathbf{w}$.
3. Result: true or false.
4. begin:
5.   client:
6.   For $w = w_1, \cdots w_f$ do
7.     $c_i \leftarrow 1$, $stag_w \leftarrow F(K_w, g_1^{\frac{1}{w}} \mod n)$
8.     for $id \in DB(w)$ do
9.       $l^* \leftarrow P(K_a, id)$, $i \leftarrow route(l^*)$.
10.      label $\leftarrow F(stag_w, c_i), e \leftarrow \text{Enc}(mpk, l^* || k_{id}, P)$
11.      xind $\leftarrow F_p(K_I, l^*)$; $z \leftarrow F_p(K_Z, g_2^{1/w} \mod n || c_i)$
12.      $y \leftarrow \text{xind} \cdot z^{-1}$; xtag $\leftarrow g^{F_p(K_X, g_3^{1/w} \mod n) \cdot \text{xind}}$
13.      LEDB$_i$[label] $\leftarrow (e, y)$,
14.      LBSIndex$_i$ $\leftarrow$ LBSIndex$_i$ $\cup$ {xtag}
15.      $c_i \leftarrow c_i + 1$
16.    end for
17.  end for
18.  Service Nodes:
19.    For node $i = 1$ to $N$
20.      store LEDB$_i$[label], LBSIndex$_i$
21.    end for
```

Fig. 3: Multi-client setup algorithm.

2) Multi-client Authorization and Search Token Generation: In our design, we assume a client submits attributes set $\mathbf{s}$ to the data owner. If this set is within the data owner's access policy $\mathbf{S}$, then the data owner will issue permitted search key sets to this client, i.e., $\mathbf{w} = (w_1, w_2, \cdots, w_S)$ with respective primer integers $(P_1, P_2, \cdots, P_S)$, and the following private keys $sk = (sk_S, K_w, K_I, K_Z, K_X, sk_{\mathbf{w}})$, where $sk_S \leftarrow KeyGen(msk, S)$ and $sk_{\mathbf{P}} = (sk_{\mathbf{P}}^{(1)}, sk_{\mathbf{P}}^{(2)}, sk_{\mathbf{P}}^{(3)})$ where $sk_{\mathbf{P}}^{(i)} = (g_i^{1/\Pi_{j=1}^n P_j} \mod n)$ for $i \in [3]$.

To generate a search token at a client side, we assume a client wishes to perform the conjunctive search with $w_1 \wedge w_2 \wedge \cdots \wedge w_m$. The least frequent keyword (assume P_1) will be chosen to generate stag $\leftarrow F(K_w, (sk_{\mathbf{P}}^{(1)})^{\Pi_{P' \in \mathbf{P} \setminus \{P_1\}} P'} \mod n) = F(K_S, g_1^{1/P_1} \mod n)$.

For each remaining keyword w_j, it will generate search token xtoken$[c, j]$ for the c-th encrypted document as following: xtoken$[c, j] \leftarrow$ $g^{F_p(K_Z, (s_{\mathbf{P}}^{(2)})^{\Pi_{P' \in \mathbf{P} \setminus \{P_1\}} P'} \mathrm{mod} n || c) \cdot F_P(K_X, (s_{\mathbf{P}}^{(3)})^{\Pi_{P' \in \mathbf{P} \setminus \{P_j\}} P'} \mathrm{mod} n)}$ $= g^{F_p(K_Z, g_2^{1/P_1} \mod n || c) \cdot F_p(K_X, g^{1/P_j} \mod n)}$.

3) Parallel Boolean Search: After generating the search token st = (stag, xtoken[1], xtoken[2], $\cdots$), the client can send the query request to servers to perform the Boolean search. In our design, we propose the parallel search, where the client can send all search tokens simultaneously to all servers. Based on the characteristic of local LEDB and LBSIndex, we can see that from local LEDB, all respective document IDs stored on a single server corresponding to the least frequency keyword can be retrieved. For each of these document IDs, if it also includes another keyword, this ID and keyword should be already indexed in the local LBSIndex. The LBSIndex serves as the test of existence of a keyword within a document

```
Multi-clients search
1. Input: $st = (stag, xtoken[1], xtoken[2], \cdots)$,
        LEDB, LBSIndex
2. Output: $R$.
3. Function: SEARCH(st, LEDB, LBSIndex).
4. Simultaneously sending $st$ to all servers 1 to N.
5.   At server $i$:
6.     $R_i \leftarrow \{\}$
7.     for $stag \in stags$ do
8.       $label \leftarrow F(stag_P, c)$;
9.       while $label \in$ LEDB do
10.        $(e, y) \leftarrow \text{LEDB}_i[label]$
11.        if $xtoken[c, j]^y \in \text{LBSIndex}_i$ for all $j$ then
12.          $R \leftarrow R \cup \{e\}$
13.        end if
14.        $c \leftarrow c + 1$; $label \leftarrow F(stag, c)$
15.  Send $R_i$ to client.
```

Fig. 4: Multi client search algorithm.

iD> Thus, from LEDB and LBSIndex, the server can check whether a document is a conjunctive result of two keywords. This allows all servers perform the parallel search independently. There, instead of passing search token from nodes 1 to N (assume total N servers), the client sends the search tokens and query request to all servers in parallel. Upon receiving the search token, all server can simultaneously perform the search operations and return the matched documents to the clients. At each server i, it calculates $label \leftarrow F(\text{stag}, c)$ by increasing c from 1 to check whether $label \in \text{LEDB}_i$ and terminates when label $\notin \text{LEDB}_i$ respected to some c values. For $label \in \text{LEDB}_i$ and $(e, y) \leftarrow \text{LEDB}_i[label]$, if $xtoken[c, j]^y \in \text{LBSIndex}_i$ for all j, the corresponding e is the conjunctive result and we take $R_i \leftarrow R_i \cup \{e\}$. After completing this operation, each server i sends R_i back to the client. The clients can integrate $R_1, R_2, \cdots, R_N$ together to get the conjunctive results. Obviously, this parallel computation can significantly accelerate the search progress. Fig. 4 shows the detailed parallel search progress.

V. Security Analysis

To show the security strength of our proposed scheme, we quantify the leakage information and show the security guarantee against attackers from adaptive servers and the clients. Recall as discussed in threat model (Section II-B), the data owner will never expose its private key to public, thus the encrypted database can be assumed secure for a passive attacker. In this section, we focus our discussion on the positive attackers who have strong ability to learn and analyze the search patterns and results patterns involved in the multi-client conjunctive queries.

We discuss the leakage profile that can be learned by adversaries and present the security definition based on the simulation-based security analysis. In distributed KV store, the encrypted database LEDB are distributed across multiples servers, thus each server only has a small portion of the database and can only learn the local search patterns and results patterns. This obviously strengthens the security guarantee comparing to the centralized server design, since it is impractical to compromise all servers to collude with each other to obtain the entire database information. This security strength will be consistently enhanced with the growing of the number of servers. When comparing with the current state-of-art work, i.e., [5], [9], the leakage profile at each server from our encryption scheme is only a small portion of that from them, since the whole database is distributed across all servers and each server only includes a small portion of the database. Even in the worse case, all servers are assumed to collude with each other and collect all leakage profiles from other servers, the leakage profile is the same as in [5], [9]. Thus, we can conclude that the leakage profile from our proposed protocol is no more than that from [5], [9], which guarantee that our scheme has higher security guarantee.

Let Π=(EDBSetup, ClientGen, TokenGen, Search) be our encryption scheme and $\mathcal{A}, \mathcal{S}$ be two efficient algorithms. In term of the quantified leakage, we present the security definition via a real experiment **Real** and an ideal experiment **Ideal** as follows:

- $\mathbf{Real}^{\mathbf{\Pi}}_{\mathcal{A}}(k)$: $\mathcal{A}(k)$ choose a database DB. The experimental runs the algorithm $\{(MK, PK, \text{LEDB}_i, \text{LBSIndex}_i) : 1 \leq i \leq N\} \leftarrow EDBSetup(1^k, DB, RDK, \mathcal{U})$ and returns $(PK, \text{LEDB}_i, \text{LBSIndex}_i)$ to $\mathcal{A}$. After that, $\mathcal{A}$ selects a set of authorized keywords (i.e., $\mathbf{w}$) for a client and then performs a polynomial number of t adaptive queries, where we assume the keyword associated with t are always within the authorized keyword set $\mathbf{w}$. To response, the experiment runs the remaining algorithm in Π (including ClientGen, TokenGen, and Search), and gives the transcript and client outputs to $\mathcal{A}$.
- $\mathbf{Ideal}^{\mathbf{\Pi}}_{\mathcal{A}}(k)$: $\mathcal{A}(1^\kappa)$ chooses a databse DB. The game initializes an empty list and a counter $i = 0$. Then the experiment runs $\{(PK, \text{LEDB}_i, \text{LBSIndex}_i) : 1 \leq i \leq N\} \leftarrow S(\mathcal{L}(DB))$ and gives $(PK, \text{LEDB}_i, \text{LBSIndex}_i)$ to $\mathcal{A}$. $\mathcal{A}$ repeatedly chooses a search t. To response, the experiment records this query as $t[i]$, increments i and gives the output to $S(\mathcal{L}(DB(t))$ for $\mathcal{A}$, where t consists of all previous queries in addition to the latest query issued by $\mathcal{A}$. Experimentally, the experiment outputs the bit that $\mathcal{A}$ returns.

Our encryption scheme Π is adaptively secure with above leakage file if for all PPT adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that $Pr[\mathbf{Real}_{\mathcal{A}}(k) = 1] - Pr[\mathbf{Ideal}_{\mathcal{A},\mathbf{S}}(\mathbf{k}) = 1] \leq negl(k)$, where $negl(k)$ is a negligible function in k.

Theorem 1: Our scheme is $\mathcal{L}-$semantically secure ($\mathcal{L}$ is the leakage function defined as before) against no-adaptive attacks if F and F_p are secure PRFs and that ABE is a CPA secure attribute-based encryption.

Proof Since the leakage profile in our scheme is no more than that from [9]. Our proof can follow the same proceed as in [9]. We omit its discussion here to conserve space.

Discussion on SSE attacks: Although the security notion of SSE explicitly defines the information learned during the search protocols, the impact of that information is not indicated. In fact, several recent studies [12], [13] show that the leakage could be exploited (e.g., frequency analysis) to compromise the confidentiality of documents and queries, especially when the adversary knows some background information about the documents and queries. An effective mitigation approach as indicated in [12] is to introduce dummy keywords and documents to obfuscate access patterns. To mitigate active injection attacks [13], SSE schemes with forward security [14] should be adopted, which prevent the servers from learning additional information from newly added documents. As future work, we will integrate the above techniques to our system.

VI. Experimental Evaluation

A. Implementation Environment

The goal of this implementation is to evaluate the efficiency of our protocol in term of data processing time and multi-client Boolean query time. We have implemented the prototype and deploy it to the Amazon Web Services (AWS). In our experiments, we create a set of AWS M4-xlarge instances, with one instance as data owner, four instances as clients, and a cluster of instances as the server nodes. For each instance, the Ubuntu server 14.04 is installed, and the configurations include 4 vcores (2.4 GHz Intel Xeon@E5-2676 v3 CPU), 16GB RAM and 40 GB SSD. The bandwidth between any two instances is 1Gbps. To achieve remote data operations among different instance, we use the software framework Apache Thrift (v0.9.3) to implement the remote procedure call (RPC). In our prototype, the OpenSSL (v1.0.2a) is used for the cryptographic build blocks, and Redis 3.2.0 is used to store data on each instance. The PRF function is implemented via calling HMAC-SHA2 and the symmetric encryption for data is implemented via AES/CBC-256.

In our experimental, we choose TPC-H/dbgen [15] to generate the required dataset, which includes a total of 10×10^6 records to represent document IDs. Different number of documents IDs within this dataset will be considered in respective performance studies. The keywords for each document varies from 100 to 10000, and will be specified in the respective performance evaluation.

B. Date Encryption and Uploading time

This time refers to both the data processing time and uploading time, where data processing time includes the data encryption time (building LEDB and LBSIndex) and uploading time is the time consumption to transmit data from data owner to servers as shown in Fig. 3.

Fig. 5 shows the processing time for the 100000 document IDs. We consider the servers' amounts increase from 1 to 8. From this figure, we can see that the processing time is significantly decreased with the increasing of server numbers. This is due to the parallelized uploading operations from data owner to multiple servers. In this figure, the processing time is only 2.65s with 8 servers comparing to 13.04s with 1 server. If the servers amounts keep increasing, the time costs for processing data will keep reduced. This demonstrates the advantages of distributed servers over a single centralized server on the data processing operations in term of data encryption and uploading.

Fig. 6 shows the time cost for the processing operations with different sizes of document IDs, which varies from 0.1×10^6 to 10×10^6. From this figure, we can see the time cost increases linearly with the sizes of the dataset. Even the dataset size is 10×10^6, the processing time is less than half minute, i.e., $28.23s$. This shows the processing operations of building local encryption structure LEDB and LBSIndex will not cost too much time, which is within the acceptable level.

C. Query Time Evaluation

To show the efficiency of query performance, we consider the number of servers increases from 1 to 8, and the size of dataset increase from 0.1×10^6 to 10×10^6. The number of document IDs that contains each keyword will increase with the growing of dataset sizes. Fig. 7 shows the time cost of Boolean search with the increasing number of servers when the dataset sizes are 0.1×10^6, 1×10^6 and 2×10^6, respectively. We assume the conjunctive results of any two keywords scale with the sizes of dataset. From this figure, we can see the query latency decreases when the servers amounts increase from 1 to 8. Especially, when the dataset size is 1×10^6, the query latency is $0.34s$ with 1 server and $0.05s$ with 8 servers. If servers' amount keeps increasing, the query latency will keep reducing. This confirms the query efficiency of distributed servers over a single centralized server, since our scheme can effectively handle queries in parallel. In this result, we can see the query time does not strictly linearly decrease with the number of servers. This is due to two reasons: 1) The connection process between client and servers costs certain time; 2) The number of matched results are different among servers and this query time is determined by the server with the maximum number of matched results. For example, when the dataset size is 0.1×10^6, the latency has only slight changes when the server' amounts increase from 3 to 8, since the connections progress dominates the latency between the client and servers.

To demonstrate the scalability of our protocol, we show the query complexity is determined by the number of matched resulted from the least frequency keyword. We set the number of document IDs that corresponding to the least frequency keyword as 100. For other query keywords, each one will be contained in the arbitrary number of documents and this number is at least 100. We consider servers number as 8. Fig. 8 shows the query time with different sizes of datasets which vary from 0.1×10^6 to 10×10^6 (number of document IDs). Note here, the matched document IDs for least frequency keywords is 100 and will not change with the size of the dataset. From this figure, we can see the Boolean query time does not increase with the sizes of the dataset. This confirms that the search complexity is independent of the dataset sizes.

D. Comparison with Interactive Method.

In this work, we advocate the non-interactive mechanisms where the data owner is not required to stay online to process clients' request. To demonstrate the advantage of non-

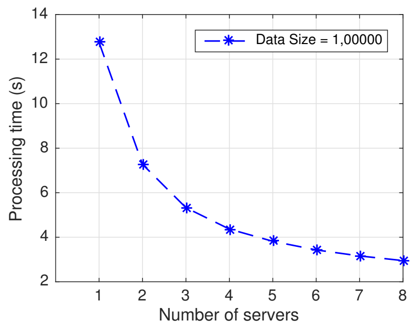

Fig. 5: The processing time with different number of servers.

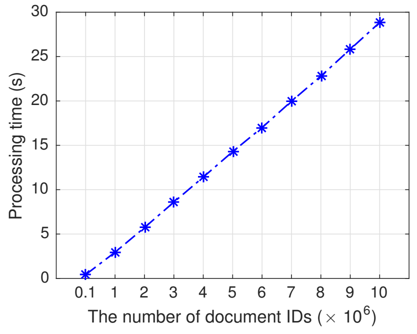

Fig. 6: The processing time with of different sizes of dataset over 8 servers.

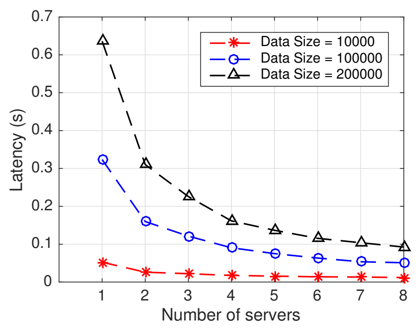

Fig. 7: The search time with different number of servers.

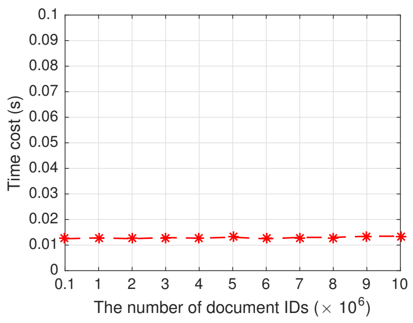

Fig. 8: The time cost with the increasing of documents that contains least frequency keyword.

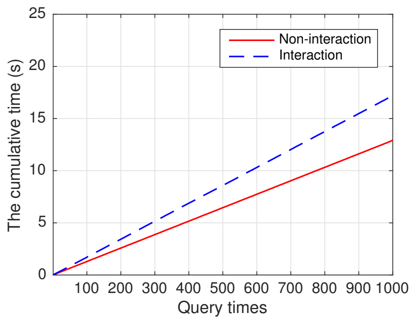

Fig. 9: The comparison of the time under interactive and non-interactive methods with 1000 queries..

interactive scheme, we compare its time consumption with that from the interactive method. The interactive method represents the scheme where the client needs to interact with the data owner for each query. We use the same setting as in Fig. 8 and perform 1000 times query. Fig. 9 shows the comparison of the cumulative time consumption between interactive and non-interactive methods under 1000 times query. From this figure, we can see that under the interactive method, the time consumption will increase faster comparing to non-interactive method. This is due to that for each query, the client needs to interact with the data owner to request search token. But under the non-interactive method, the client only needs to interact with the data owner one time. After one time interaction, the client can alway generate the search token by itself without the interaction with the data owner any more. This saves a huge amount of time and improve the whole search speed. From Fig. 9, we can see that if a client performs 1000 times query, the total time consumption is 17.20s under interactive method, while only 12.90s under the non-interactive method. The interactive method will cause 4.3s latency compared to the non-interactive method. If a client continues to perform more queries, this latency will also increase, which thus cause more delay. Note here, these results are from the excellent network environment where the bandwidth between the client and data owner is 1Gbps. If the network environment is worse (low bandwidth), the interaction will cause even more delay and slow down the search speed. This demonstrates the advantages of our method over the interactive method.

VII. Related Works

Searchable Symmetric Encryption: Searchable symmetric encryption (SSE) [1], [2] has become a prominent cryptographic methodology for privacy-preserving data applications in cloud computing, with the advantage of supporting efficient search over encrypted data. There were many active efforts [2], [14], [16], [17] to design specific SSE schemes from the perspectives of security, efficiency, and functionality. Since using single keyword search SSE schemes for boolean queries neither scales well nor guarantees minimized leakage, dedicated schemes for boolean queries are designed [4], [5]. In [5], Cash *et al.* proposed the first scheme that achieves sublinear search complexity for conjunctive queries, and reduces the leakage only reflecting the access pattern of conjunctive keywords. After that, Faber *et al.* extended the above scheme to enable secure range, substring, and wildcard queries [18]. To further improve efficiency, Kamara and Moataz recently proposed a scheme that achieves sublinear search complexity for disjunctive and arbitrary boolean queries [4].

Most SSE schemes consider a simplified setting which includes only one client and one server. As recognized, directly deploying those schemes to realistic applications sometimes is not sufficient to serve the needs of involving multiple clients and a cluster of distributed servers.

Multi-client Access in Searchable Encryption: Although asymmetric key based searchable encryption schemes [7], [19] (just to list a few) facilitates the management of multi-client access privileges with privacy preservation, those schemes need to compute over the all encrypted documents, which is not scalable for large datasets. Therefore, we are interested in symmetric key based schemes. In [2], Curtmola *et al.* gave the first construction for multi-client SSE based on broadcast encryption. However, their scheme only enables a basic access control policy such that as long as a client is granted access, it is allowed to generate arbitrary keyword queries.

To enforce a more strict and refined access policies for different clients, Jarecki *et al.* leveraged oblivious PRF for the generation of keyword tokens [6]. As a result, only authorized keywords can be queried. In addition, they apply the proposed technique to Cash *et al.*'s scheme [5] for boolean queries. One potential performance issue of this work is that the data owner is required to always stay online to interact with the client to run the oblivious PRF protocol. To address this issue, Sun *et al.* proposed a non-interactive scheme, where a client only needs to communicate with the data owner one time to obtain the necessary search keys for authorized keywords. With the search keys, the client can perform any boolean queries within a specific policy without the interaction with the data owner. Unfortunately, those schemes focus on the construction of theoretical primitives, and thus provide the practitioners little intuition about the integration of production systems and these cryptographic primitives. Our design aims to identify and bridge the gaps between them.

Encrypted Data Management Systems: The encrypted database systems [20]–[22] have been implemented to support a large portion of database queries over encrypted data. Among them, a system named BlindSeer [20] supports practical and arbitrary boolean queries. The technique is to encode a boolean query in a Bloom filter and query an encrypted Bloom filter index via secure function evaluation protocols. We note that those systems are designed for conventional relation database systems. Recently, Yuan *et al.* proposed a distributed and encrypted key-value store [8]. This data store provides an encrypted local index framework to facilitate secure queries in parallel. Afterward, they further realize range-match and exact match queries under the index framework [23]. But their design does not support practical boolean queries, where each boolean query is still required to be split into individual queries for each query attribute.

VIII. Conclusions

In this paper, a new secure multi-client Boolean search scheme is designed to provide guarantees of data confidentiality and satisfy the search efficiency. By leveraging distributed index framework, this design is distributed in nature to support the system scalability needs. To keep data confidential and privacy, we designed a new scheme to generate the local encrypted index LEBD and LBSIndex which are distributed across multiple servers. Based on this scheme, we show how to construct encryption structure, generate search token, and search in parallel to achieve the efficient Boolean search. On the other hand, to authorize multi-client access for data owner's data, we advocate the non-interactive scheme where the data owner is not required to stay online to process clients' access request. To demonstrate the security and privacy guarantees, we provide the formal security analysis to illustrate that our protocol can achieve strong security guarantee comparing to the existing works. Finally, a system prototype is designed and deployed to Amazon EC2 to evaluate and demonstrate the effectiveness and efficiency of our proposed scheme.

References

[1] D.X. Song, D. Wagner, and A. Perrig, "A practcal techniques for searches on encrypted data," in *IEEE S&P*, 2000.

[2] R. Curtmola, J. Garay, S. Komara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. ACM CCS*, 2006.

[3] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography*, 2013.

[4] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Eurocrypt*, 2017.

[5] D. Cash, S. Jarecki, C.S. Jutla, H. Krawczyk, M-C. Rosu, M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *Proc. CRYPTO*, 2013.

[6] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *Proc. ACM CCS*, 2013.

[7] F. Bao, R. Deng, X. Ding, Y. Yang, "Private query on encrypted data in multi-user settings," in *ISPEC*, 2008.

[8] X. Yuan, X. Wang, C. Wang, C. Chen, and J. Lin, "Building an encrypted, distributed, and searchable key-value store," in *Proc. ACM ASIACCS*, 2013.

[9] S.F. Sun, J.K. Liu, A. Sakzad, R. Steinfeld, and T.H. Yuen, "An efficient non-interactive multi-client searchable encryption with support for boolean queries," in *Proc. ESORICS*, 2016.

[10] M. Naveed, S. Kamara, and C.V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. ACM CCS*, 2015.

[11] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart,, "Leakage-abuse attacks against searchable encryption," in *Proc. ACM CCS*, 2015.

[12] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart , "Leakage-abuse attacks against searchable encryption," in *ACM CCS*, 2015.

[13] Y. Zhang, J. Katz, and C. Papamanthou, " All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *USENIX Security*, 2016.

[14] R. Bost, "Sophos: Forward Secure Searchable Encryption," in *ACM CCS*, 2016.

[15] Transaction Processing Performance Council, "TPC Benchmark H (Decision Support) Standard Specification," in *TPC*, 2002.

[16] F. Hahn and F. Kerschbaum, "Searchable Encryption with Secure and Efficient Updates," in *CCS*, 2014.

[17] D. Cash, J. Jaeger, S. jareck, and C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proc. ACM NDSS*, 2014.

[18] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *ESORICS*, 2015.

[19] C. Dong, C. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," in *DAS*, 2008.

[20] V. Pappas, B. Vo, F. Krell, S. Choi, V. Kolesnikov, A. Keromytis, and T. Malkin., "Blind Seer: A scalable private DBMS," in *Proc. IEEE S&P*, 2014.

[21] R. Poddar, T. Boelter, and R. A. Popa, "Arx: A strongly encrypted database system," in *Cryptology ePrint Archive, Report 2016/591*, 2016.

[22] R.A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *Proc. ACM SOSP*, 2011.

[23] X. Yuan, Y. Guo, X. Wang, C. Wang, B. Li, and X. Jia, "EncKV: An encrypted key-value store with rich queries," in *Proc. ACM. ASIACCS*, 2017.