Enabling Encrypted Boolean Queries in Geographically Distributed Databases

Xu Yuan, *Member, IEEE*, Xingliang Yuan, *Member, IEEE*, Yihe Zhang, *Student Member, IEEE*, Baochun Li, *Fellow, IEEE*, and Cong Wang, *Senior Member, IEEE*

Abstract—The persistent growth of big data applications has being raising new challenges in managing large volumes of datasets with high scalability, confidentiality protection, and flexible types of search queries. In this paper, we propose a secure design to disassemble the private dataset with the aim to store them across geographically distributed servers while supporting secure multi-client Boolean queries. In this design, the data owner encrypts the private database with the searchable index attributes. The encrypted dataset will be disassembled and distributed evenly across multiple servers by leveraging the property of a distributed index framework. By constructing an encryption structure, generating search tokens, and enabling parallel query, we show how the proposed design performs the secure while efficient Boolean search. These queries are not only limited to those initiated by the data owner but also can be extended to support multiple authorized clients, where each client is allowed to access a necessary part of the private database. In this stage, we advocate a non-interactive authorization scheme where data owner is not required to stay online to process the query request. Moreover, the query operation can be executed in parallel, which significantly improves the search efficiency. We formally characterize the leakage profile, which allow us to follow the existing security analysis method 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. Through experimental results, we demonstrate the effectiveness of our protocol in terms of data outsourcing time and Boolean query time.

Index Terms—Searchable symmetric encryption, multi-client data access, key-value stores, Boolean query

1 Introduction

The last decade has witnessed a significant increase in the volume of data generated every day, reaching an order of exabytes [1]. Applications of such large volume data have been extended into many areas, such as social networking, e-health systems, and smart grid systems, and others. It is becoming increasingly common for data to be hosted off-site, especially with the rise of cloud computing for use. Despite this, the need for storing such large volumes of data in the cloud has raised new challenges in 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 may protect data confidentiality, it does not readily support textual search functions over encrypted domain. 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., [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]). These works are limited either on a single centralized server or simple data operations (e.g., single keyword search), and do not well address the new requirements on storage scalability, security and privacy guarantee, as well as flexible data retrieval to serve the 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) [12], [13]) to encrypt and distribute data evenly across multiple servers. To keep data privacy, 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. Moreover, these operations are not only limited to those initiated by the data owner, but also should be capable of supporting multiple clients data access, where authorized clients can access the private database owned by a data owner. In particular, we study how to enable the secure and efficient Boolean search in a multi-client setting to fulfill the needs of real-world data applications.

Boolean search has been intensively applied in the large dataset. For example, in hospital database systems, if a

- *X. Yuan and Y. Zhang are with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70504. E-mail: {xu.yuan, yihe.zhang1}@louisiana.edu.*
- *X. Yuan is with the Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia. E-mail: xingliang.yuan@monash.edu.*
- *B. Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S, Canada. E-mail: bli@ece.toronto.edu.*
- *C. Wang is with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. E-mail: congwang@cityu.edu.hk.*

Manuscript received 17 Dec. 2018; revised 18 June 2019; accepted 30 Aug. 2019. Date of publication 12 Sept. 2019; date of current version 10 Jan. 2020. (Corresponding author: Xu Yuan.)
Recommended for acceptance by S. Chen.
Digital Object Identifier no. 10.1109/TPDS.2019.2940945

doctor requires investigating all male patients with the diabetes clinical trial, she will perform the conjunctive search for keywords "male" and "diabetes". In the encrypted domain, Boolean search can be realized to search each individual keyword. By doing so, prior SSE schemes for single keyword search [3], [4], [10] can directly be used. However, this treatment will introduce two issues. The first is that the search time scales with the number of keywords in a Boolean query. If some keywords match a large portion of documents in a dataset, they may incur a long query latency. The other is that the server learns all the matched documents of each individual keyword, even some documents are not in the final query results, and are not expected to be revealed. Although there are some recent efforts [8], [14] working on to address these issues, how to apply them into an encrypted and distributed data store remains an open problem.

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 data. One solution is to use broadcast encryption as introduced in [4], but it is not scalable for a large number of users while supporting single keyword search only. Recent solutions in [5], [15] support Boolean search in specific access policies, but they ask a client to communicate with data owner for the 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 [8] 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 [11], [16], so as to disassemble the entire database and store them across geographically distributed servers. 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 which can be applied for Boolean queries. Such local structures enable the data owner or clients to perform the Boolean queries in parallel among all servers.

In a multi-client setting, to control clients' access privileges of Boolean queries, we further integrate the latest multi-client SSE scheme [6] into our design, and advocate this non-interactive scheme, where a client only needs to interact with the data owner one time 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 [17]. 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 our designed Boolean search, and show that the information leakage is no more than that from the current state-of-the-art works, i.e., [6], [8]. 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. Through experimental results, we validate the correctness and demonstrate the effectiveness of our scheme.

The main contributions are summarized as follows:

- We design a secure encryption scheme to support Boolean search based on a distributed index framework. The original database is disassembled into local encrypted database and search structures which can be stored evenly across 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. Thus, the search efficiency is significantly improved. On the other hand, the communication overhead and workload on each server can be substantially reduced when comparing to a single centralized server since each server can only be responsible of processing a small portion of data, which eventually improve the 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 one time to obtain the necessary search keys. In this process, both data owner's data and clients' search information can be well protected. With the requested keys, a client can always generate the search token by itself to perform Boolean queries without interacting with data owner.
- We characterize and analyze the leakage profile from both the encryption structure and query process, which allow us to follow the existing formal security analysis to demonstrate that our system can guarantee data confidentiality and query privacy. 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 Amazon EC2 to validate our design. The experimental results show that both the data outsourcing 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 2, we introduce the system model for this paper. Section 3 discusses the necessity and importance of Boolean search and multi-client query over distributed KV store. Section 4 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

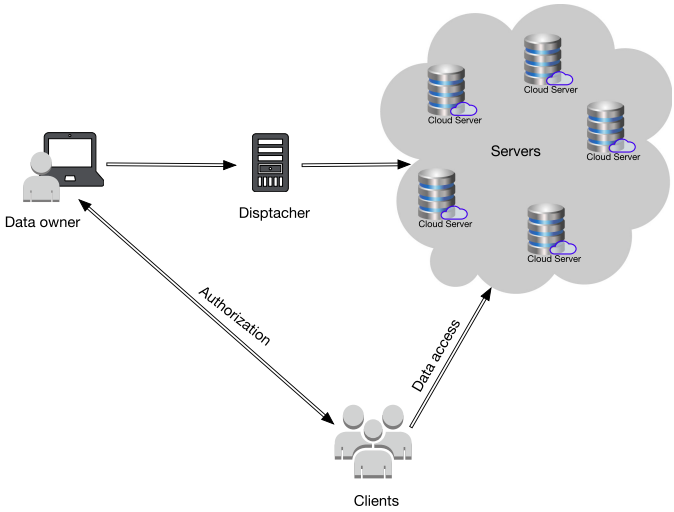

Fig. 1. The system architecture.

are generated, and parallel queries are performed. In the end, we show how to extend our design into general queries. In Section 6, we characterize the leakage profiles and discuss the maximum leakage incurred to adversaries and clients in the encryption scheme and query process. In Section 7, we present the implementation and performance evaluation of our protocol. Section 8 discusses related work and Section 9 concludes this paper.

2 System Model

2.1 Overview

In this section, we describe our system architecture that we rely on to support secure multi-client data access and Boolean search. As shown in Fig. 1, data owner, clients, dispatcher, and a number of distributed public cloud servers are the main entities in this system. The former two entities are individuals or enterprise users while the following two are public cloud service nodes. Specifically, the data owner is a user who has a private dataset and wishes to store the encrypted version of this dataset into public cloud service nodes. The role of a dispatcher is to disassemble the encrypted dataset and route them evenly across all distributed servers. In this architecture, there exist some clients who wish to access the data owner's database on cloud but have to request the authorization.

In our setting, we assume the information of the private dataset from the data owner includes the documents, document IDs, and keywords. The data owner aims to store such a dataset on the remote geographically distributed servers while maintaining its confidentiality and search capability. 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 each document using the standard algorithm (i.e., AES), while building the searchable index with document ID to enable the query operation. To disassemble the dataset so as to distributively store them on multiple distributed servers, the data owner needs to build the local encryption database and local searchable index respected to each individual server, and then sends them to the dispatcher so as to decide which server will be routed to. In addition to perform the search with document ID, we also enable the second attributes search, i.e., keyword search, to enhance the data owner's search ability on the outsourced data. That is, for data owner, it can perform the search either on document identifiers or keywords to find the matched documents. In our model, we assume the data access shall not only be limited to the data owner but also can be extended to other clients by authorizing their access. Due to the privacy concern from the data owner side, the authorized clients are only allowed to perform keyword search.

In this work, we focus our study on enabling the encrypted dataset on cloud servers to support the Boolean search (i.e., conjunctive, disjunctive, and negate) that are submitted from clients. We first use conjunction query as an example to present our design, i.e., $w_1 \wedge w_2 \wedge \cdots \wedge w_m$. That is, one client submits the conjunctive search request $w_1 \wedge w_2 \wedge \cdots \wedge w_m$ to cloud servers. Upon receiving this request, cloud servers perform the query and return all matched documents including keywords $w_1, w_2, \cdots, w_m$. Later, we show that our protocol can be easily adapted to any other form of Boolean search expressions.

2.2 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 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 while analyzing both the query and result patterns. In this paper, we do not consider the case where attackers can access the background information about the queries and database, including database structure, statistic information of database, the relationships of different queries, and others. Thus, inference attack [18] and leakage-abuse attack [19] will not be included here. Also, we do not consider the case where malicious attackers can modify or delete the database intentionally.

2.3 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 as proposed in [11], 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

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.

id_i. The documents identifiers **id** are protected with pseudo-random function PRF, and the documents **f** are encrypted with the symmetric encryption algorithm. Then the entry of KV store is defined as:

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

where K_{id} and K_f are the private keys. The dispatcher can use consistent hashing to [20] find the target server with the input of l^*. With consistent hashing, all encrypted database can be disassembled and distributively stored across multiple nodes with a balanced distribution.

3 Boolean Search and Multi-client Query over Distributed KV Store

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

3.1 Boolean Search

According to our system model, one straightforward method of executing Boolean search is to replace the conjunctive search into m single keywords search while returning a set of matched documents for each individual keyword. Upon receiving all matched document sets, the client performs the intersection operation to find the common documents among these sets. That is, the client performs search for each keyword w_i and returns a set of matched encrypted documents (denoted as set F_i) to the client. With all returned encrypted document sets F_i where $i = 1 \cdots m$, the client finds the intersection among $F_1, F_2, \cdots, F_m$, i.e., $F_1 \cap F_2 \cdots \cap F_m$, to find the conjunctive search results. This method is simple but inefficient since it returns too many redundancy documents. The complexity is the sum of all matched documents for searched keywords. Especially, if one keyword is included in all documents, the entire 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 searched keyword.

To overcome such weakness, one approach is to enable servers to perform the conjunctive search and only return intersection results to the client. Fig. 2 shows one plausible solution of the conjunctive search by extending the results of [11]. By observing the data structure of KV store in [11], 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^* to see if l^* is the conjunctive search result. In one word, the client can submit all keywords query requests to servers, and servers can perform the intersection operation for (α, β) sets to find the matched document indexes. Through this way, the servers need only to return the matched conjunctive search results to the clients.

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

3.2 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 entire databases, or even modify it. In this situation, the data owner will lose its control of database, which definitely is not expected to 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 search token and sends back to the client. With the search token, clients can perform the query operation to cloud servers with the obtained token from data owner. The main weakness of this solution is that it requires data owner to stay online to process per-query and generate query token responding to the client's request. On the other hand, the query information from the client will be learnt by the data owner.

With aforementioned discussions, we conclude that, to support multi-client query and Boolean search, a more secure and efficient scheme is needed.

4 The Proposed Construction

In this section, we present our design of disassembling encrypted dataset into distributed servers while supporting multi-client Boolean queries. First, we introduce how to deploy a 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 multi-client settings.

We rely on the idea from [8] 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 pair, 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 both the document and a keyword of this document while the value indicates the existence.

To enable authorized queries, we customize a secure non-interactive scheme proposed in [6] into our design to enable the multi-client query over distributed KV store. 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 other effective data operations (e.g., modify database) rather than only performing the permitted query operations.

4.1 Build Encrypted Database

To enhance search efficiency, we consider the search respected to (keyword, document ID) pairs instead of (keyword, document) pairs. As the server stores both documents and associated IDs, once an ID is located, the corresponding document can be quickly retrieved. Thus, the search for a document ID is the same as for a document itself.

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., $(P_1, P_2, \cdots, P_m) \leftarrow (w_1, w_2, \cdots, w_m)$. In the following section, we will use P to represent keyword for ease of discussion.

To disassemble the encrypted database and store distributively across multiple servers, it is important to build local index sets rather than the global index as that has been proposed in [8]. Particularly, we aim to build both the local encrypted database and local boolean search index (LBSIndex) corresponding to each individual server. Here, 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

Secure Put
1. **Input:** Private key $\mathbf{K}$, DB, $\mathbf{P} \leftarrow \mathbf{w}$.
2. **Result:** *true or false*.
3. **begin:**
4. client:
5. For $P = P_1, \cdots P_f$ do
6. $c_j \leftarrow 1$, $stag_P \leftarrow F(K_P, g_1^{\frac{1}{P}} \mod n)$
7. for $id \in DB(w)$ do
8. $l^* \leftarrow PRF(K_w, id)$, $i \leftarrow route(l^*)$.
9. label $\leftarrow F(stag_P, c_i)$,
10. $e \leftarrow Enc(l^* || k_{id})$,
11. xind $\leftarrow F_p(K_I, l^*)$; $z \leftarrow F_p(K_Z, g_2^{1/P} \mod n || c_i)$
12. $y \leftarrow$ xind $\cdot z^{-1}$; xtag $\leftarrow g^{F_p(K_X, g_3^{1/P} \mod n) \cdot \text{xind}}$
13. LEDB$_i$[label] $\leftarrow (e, y)$,
14. LBSIndex$_i \leftarrow$ LBSIndex$_i \cup$ {xtag}
15. $c_j \leftarrow c_j + 1$
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.

keys K_w for a PRF PRF, 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 $MK = (p, q, K_w, K_I, K_Z, K_X, g_1, g_2, g_3)$ and the corresponding system public key $PK = (n, g)$, where $n = pq, g \xleftarrow{\$} G$ and $g_i \xleftarrow{\$} Z_n^*$ for $i \in \{1, 2, 3\}$. Here, G is a cyclic group of prime order p.

For each server i, we initiate the LEDB_i and LBSIndex_i as the empty sets. Regarding each document, we store the key-value pair as l^* and P, where l^* corresponds to the associated document ID and P corresponds to a keyword. As we have mentioned, once the document ID is located, the document can be quickly retrieved. Thus, the l^* should be well protected as well as the relationships between l^* and keyword P, which are the main objectives of our design. Since each P will be contained in multiple documents, we increment an integer value c from 1 regarding to each document. To protect the l^* and keyword relationships, we build the LEDB and LBSIndex with the following operation.

- We build local LEDB on each server i in the following format: $e \leftarrow Enc(l^* || k_{id})$, $\text{xind} \leftarrow F_p(K_I, l^*)$; and $z \leftarrow F_p(K_Z, g_2^{1/P} \bmod n || c_j)$, 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 $stag_P \leftarrow F(K_P, g_1^{\frac{1}{P}} \bmod n)$ and $label \leftarrow F(stag_P, c_j)$. Then, we add (e, y) to LEDB_i, i.e., $\text{LEDB}_i[label] \leftarrow (e, y)$.
- To build LBSIndex, we set $\text{xtag} \leftarrow g^{F_p(K_X, g_3^{1/P} \bmod n) \cdot \text{xind}}$ and add xtag into LBSIndex, i.e., $\text{LBSIndex}_i \leftarrow \text{LBSIndex}_i \cup \{\text{xtag}\}$ respected to the server i.

This progress continues till all keywords and their corresponding document IDs are added into LEDB and LBSIndex. To distribute LEDB and LBSIndex, the dispatcher will decide the target server (based on l^*) that will be routed to for the encrypted keyword and l^*, so we have $i \leftarrow route(l^*)$. 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.

The advantages in design of local LEDB and LBSIndex manifest two aspects. First, with local LEDB, we can retrieve the document IDs stored on a single server corresponding to a keyword. If these document IDs also include another keyword, then the document ID and keyword should be already indexed in the local LBSIndex. This allows a server to check if a document is a conjunctive result of two keywords without any information from other servers and enables the search can be done independently among all servers. Second, the client can store the information c_i corresponding to each keyword on each server. Once a new document is added to the database (i.e., update operation), the new index entry can be immediately built by increasing the value of c_i that respects to a keyword on server i (assuming new document will be routed to server i) and added to LEDB_i and LBSIndex_i without rebuilding the entire LEDB and LBSIndex.

$$\begin{aligned}\text{xtoken}[c,j] &\leftarrow g^{F_p\left(K_Z,\left(s_{\mathbf{P}}^{(2)}\right)^{\Pi_{P'\in\mathbf{P}\setminus\{P_1\}}P'}\bmod n\|c\right)\cdot F_p\left(K_X,\left(s_{\mathbf{P}}^{(3)}\right)^{\Pi_{P'\in\mathbf{P}\setminus\{P_j\}}P'}\bmod n\right)}\\ &= g^{F_p\left(K_Z,g_2^{1/P_1}\bmod n\|c\right)\cdot F_p\left(K_X,g^{1/P_j}\bmod n\right)}.\end{aligned} \tag{2}$$

$$\begin{aligned}\text{xtoken}[c,j]^y &= \left(g^{F_p(K_Z,g_2^{1/P_1}\bmod n)\cdot F_p(K_X,g_3^{1/P_j}\bmod n)}\right)^{\text{xind}\cdot z^{-1}}\\ &= \left(g^{F_p(K_Z,g_2^{\frac{1}{P_1}}\bmod n)\cdot F_p(K_X,g_3^{\frac{1}{P_j}}\bmod n)}\right)^{\text{xind}\cdot\left(F_p(K_Z,g_2^{\frac{1}{P_1}}\bmod n\|c_j)^{-1}\right)}\\ &= g^{F_p\left(K_X,g_3^{1/P_j}\bmod n\right)\cdot\text{xind}}.\end{aligned} \tag{3}$$

To further enforce the access control policies on retrieved documents, we adapt ABE [21] for document encryption. Each document encryption key can be secured by ABE encryption. Only with authorized attributes, the user is able to decrypt the key ciphertext and recover the document from the search result. The ABE consists of the following algorithms ABE.Setup, ABE.KeyGen, ABE.Enc, and ABE.Dec. ABE.Setup takes the security parameter κ as input and outputs the public parameters and a master secret key. ABE.KeyGen takes a set of access attributes S, the master key, the public parameters as the input, and then output a decryption key for document. ABE.Enc takes a document, an access structure $\mathcal{A}$ and the public parameters as input and outputs the ciphertext such that only a user processing a set of attributes satisfying the access structure $\mathcal{A}$ will be able to decrypt the message. ABE.Dec takes the input of a ciphertext containing an access policy and a private key which is associated with a set of attribute S, and recovers the message if $S \in \mathcal{A}$.

4.2 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. Here, the access policy means the data owner allows this client to access the request dataset.

In our design, we assume a client submits a set of search keywords include $\mathbf{w} = (w_1, w_2, \cdots, w_m)$ with respective primer integers $(P_1, P_2, \cdots, P_m)$, and the following private keys $sk = (K_w, K_I, K_Z, K_X, sk_{\mathbf{P}})$, where $sk_{\mathbf{P}} = (sk_{\mathbf{P}}^{(1)}, sk_{\mathbf{P}}^{(2)}, sk_{\mathbf{P}}^{(3)})$ while $sk_{\mathbf{P}}^{(i)} = (g_i^{1/\Pi_{j=1}^{n}P_j} \bmod n)$ for $i \in \{1,2,3\}$.

Query Token Generation

1. **Input:** $sk, \mathbf{P} \leftarrow \mathbf{w}$
2. **Output:** st.
3. **Function:** TokenGEN(st, **P**).
4. $st, xtoken \leftarrow \emptyset, \bar{s} \leftarrow \emptyset$
5. $\hat{s} \leftarrow \hat{s} \cup \{P_1\}$
6. $\mathbf{x} \leftarrow \mathbf{P} \backslash \bar{s}$
7. $stag \leftarrow F(K_P, (sk_{\mathbf{P}}^{(1)})^{\Pi_{P'\in\mathbf{P}\setminus\{P_1\}}P'} \bmod n)$ $= F(K_P, g_1^{1/P_1} \bmod n)$
8. for $c = 1, 2, \cdots$ until the server stops
9. for each keyword in **x**, i.e., $j = 2, \cdots, m$
10. calculating $xtoken[c, j]$ based on Eqn. (2).
11. end for
12. end for
13. $st \leftarrow (stag, xtoken)$
14. return st

Fig. 4. Token generation algorithm.

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 $\text{stag} \leftarrow F(K_P, (sk_{\mathbf{P}}^{(1)})^{\Pi_{P'\in\mathbf{P}\setminus\{P_1\}}P'} \bmod n) = F(K_P, g_1^{1/P_1} \bmod n)$. For each remaining keyword w_j, it will generate search token $\text{xtoken}[c, j]$ for the c-th encrypted document as shown in Eqn. (2). Fig. 4 depicts the search token generation procedure.

4.3 Parallel Boolean Search

After generating the search token $st = (\text{stag}, \text{xtoken}[1], xtoken[2], \cdots)$, the client can send the query request to servers and perform the Boolean search. In our design, we enable the parallel Boolean search, where a client can send all search tokens simultaneously to all servers. Based on the characteristic of local LEDB and LBSIndex, we can see that with local LEDB, all respective document IDs stored on a single server corresponding to the least frequent 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 ID. Thus, from LEDB and LBSIndex, the server can check whether a document is a conjunctive result of two keywords. This allows all servers to perform the parallel search independently. Therefore, 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 servers 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. We add e to the conjunctive set R_i, i.e., $R_i \leftarrow R_i \cup \{e\}$. After completing

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. 5. Multi client search algorithm.

this operation, each server i sends R_i back to the client. The clients can integrate $R_1, R_2, \ldots, R_N$ together to get the conjunctive results. Obviously, this parallel computation can significantly accelerate the search progress. Fig. 5 depicted the parallel search procedure.

4.4 Correctness

The correctness of our construction can be verified as shown in Eqn. (3). Based on the construction of XSet in Fig. 3, we can see that if $g^{F_p(K_X, g_3^{1/P_j} \bmod n)\cdot \text{xind}} \in XSet_i$, then the keyword P_j (i.e., w_j) is included in the respected document l^*, according to xind.

5 Extensions to General Queries

The aforementioned section has presented the design of secure multi-client conjunctive query. Following the similar token, we can easily extend our protocol to support any form of Boolean queries, including disjunctive query, conjunctive with negated terms, and Boolean expressions in searchable normal form. We now show that our protocol can be readily adapted to support these forms of Boolean queries.

Disjunctive Query. To support disjunctive query, the search token generation and query process are different from that in the conjunctive query. In the token generation stage, the client needs to generate both stag and respected xtoken for all keywords rathe than only for the least frequent keyword in conjunctive query. The generation process can follow the similar method as shown in Section. 4.2. To perform the query, each server i will start from the most frequent keyword and calculate $label \leftarrow F(\text{stag}, c)$ by increasing c from 1 by checking whether $label \in \text{LEDB}_i$ and terminate when $label \notin \text{LEDB}_i$ respected to some c value. The server saves the encrypted data e and put into result set R_i, i.e., $R_i \leftarrow R_i \cup \{e\}$. Then, the server selects the second most frequent keyword and calculates $label \leftarrow F(stag, c)$ by increasing c from 1 to check whether $label \in \text{LEDB}_i$ and terminating once $label \notin \text{LEDB}_i$ respected to some c values. For $label \in \text{LEDB}_i$ and $(e, y) \leftarrow \text{LEDB}_i[label]$, if $xtoken[c, 1]^y \notin \text{LBSIndex}_i$, then $R_i \leftarrow R_i \cup \{e\}$. After that, the server selects the third most frequent keyword and find the associated (e, y). If $xtoken[c, 1] \notin \text{LBSIndex}_i$ and $xtoken[c, 2] \notin \text{LBSIndex}_i$, we add e to set R_i, i.e., $R_i \leftarrow R_i \cup \{e\}$. The server can continue above progress till the last keyword, which will check $xtoken[c, j]$ for all $1 \leq j \leq m - 1$ (m is the number of keywords). If all of them are not in LBSIndex_i, we add e to R, i.e., $R_i \leftarrow R_i \cup \{e\}$. After finishing these operations, all servers will send their result sets (i.e., R_i) to client and the client unions them together to obtain the set of disjunctive results.

Conjunctive with Negated Terms. We refer negated term query as that the searched results do not contain a given keyword. Here, we study the conjunctive search with some negated terms where at least one non-negated term exists. To perform the query, the client chooses one of the non-negated terms as the terms **stag** and the remaining as xterm (i.e., xtoken). Both stag and xtoken will be sent to servers. Upon receiving them, each server i finds the matched results and then computes whether $xtoken^y$ belong to XSet_i or not. If $xtoken[c, j]^y$ belongs to XSet_i for any j on a server i, this server will broadcast a "drop" message to all servers. After receiving the "drop" message, each server will remove the associated results from R_i. Note here, for negated query, additional communication overhead will be generated among servers. But comparing to [8], the query efficiency is significantly improved due to the parallel query operations.

Boolean Expressions in Searchable Normal Form. Based on the aforementioned discussions for conjunctive query, disjunctive query, and negated query, we can easily adapted our protocol to support any expression of Boolean queries, such as "$w_1 \wedge \theta(w_2, w_3, \cdots, w_m)$", where θ is an arbitrary combination of Boolean expressions, i.e., conjunctive, disjunctive, and negated. To perform the query, the client can calculate the necessitated stag and xtoken for $w_1, w_2, \cdots w_m$ based the Boolean categories as discussed before. We take the Boolean query $w_1 \wedge w_2 \vee w_3$ as an example. The client will generate stag for w_1 while xtoken for w_2 and w_3 and send to servers. At each server i, it first finds the (e, y) for stag, and checks if $xtoken[c, 2]^y$ or $xtoken[c, 3]^y \in \text{XSet}_i$. If there is any one of $xtoken[c, 2]^y$ and $xtoken[c, 3]^y$ belonging to XSet_i, then the server adds e to R_i, i,e., $R_i \leftarrow R_i \cup \{e\}$. In the end, each server will send its R_i to client. By putting all R_i together, the client gets the matched results for Boolean search of $w_1 \wedge w_2 \vee w_3$. By following the similar design, any Boolean query expression can be achieved. We omit their discussions here to conserve space.

6 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 2.2), 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 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-the-art work, i.e., [6], [8], the leakage profile at each server from our encryption scheme is only a small portion of that from them, since the entire 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 [6], [8]. Thus, we can conclude that the leakage profile from our proposed protocol is no more than that from [6], [8]. This guarantees that our scheme has a higher security level.

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:

- $\mathbf{Real}_{\mathcal{A}}^{\Pi}(k)$: $\mathcal{A}(k)$ chooses a database DB. The experiment runs the algorithm $\{(MK, PK, \mathrm{LEDB}_i, \mathrm{LBSIndex}_i) : 1 \leq i \leq N\} \leftarrow EDBSetup(1^k, DB, RDK, \mathcal{U})$ and returns $(PK, \mathrm{LEDB}_i, \mathrm{LBSIndex}_i)$ 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 $\mathcal{A}$.
- $\mathbf{Ideal}_{\mathcal{A}}^{\Pi}(k)$: $\mathcal{A}(1^{\kappa})$ chooses a databse DB. The game initializes an empty list and a counter $i = 0$. Then the experiment runs $\{(PK, \mathrm{LEDB}_i, \mathrm{LBSIndex}_i) : 1 \leq i \leq N\} \leftarrow S(\mathcal{L}(DB))$ and gives $(PK, \mathrm{LEDB}_i, \mathrm{LBSIndex}_i)$ 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 [6]. Our proof follows the same proceed as in [6]. 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 [19], [22] 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 [19] is to introduce dummy keywords and documents to add a protection to access patterns. For the injection attack [22], SSE schemes with forward security [23] should be adopted, which prevents from learning additional information from newly added documents.

7 Experimental Evaluation

7.1 Prototype Implementation

In this section, we design the system prototype to implement our protocol as proposed in Section 4. Fig. 6 shows the implementation of our system prototype. In our implementation, only three entries are included, which are data owner, clients and servers. The dispatcher is integrated into the data owner to simplify our design. In each entry, we use blocks to represent the operation modules. At the data owner side, the encrypted database is generated under the *Encryption* module and disassembled by the *Consistent Hashing* module, which is based on the algorithm in Fig. 3. After generating the local encrypted index sets LEDB and LBSIndex, the data owner can send them to the respective servers through multi-thread module. All encryption keys are stored in *KeyStorage* module. If the search request from a client is authorized, the *KeyGen* module will be turned on to generate the search keys for this client. Note the data owner needs to generate the search keys based on client's request rather than retrieving the encryption keys, so the simple Socket call operation for remote data retrieval cannot meet this need. Hence, we employ Apache Thrift as a stack to achieve the remote call procedure between clients and data owner.

Each client has four modules, i.e., *KeyRequest*, *GenToken*, *Parallel search* and *Decryption*. The *KeyRequest* module requires clients interacting with the data owner only one time to get the necessary search keys. After that, it can always call *GenToken* to generate the search token with any search within the specified policy. The search token is sent to all servers simultaneously through the *Parallel search* module. Here, the Apache Thrift is also employed to implement the remote procedure call between clients and servers.

At each server, after receiving the search request, it starts the *Computation* module to perform the necessary computation and retrieve the encrypted value (e, y). This (e, y) and search token will be passed to the *Check* module to perform the test operation as shown in Fig. 5. To accelerate this operation, we employ Redis.EXIST() to test the existence of each xtoken in LBSIndex. For Redis.EXIST() function, as the server only needs to check a key's existence without accessing its value, the time cost is extremely low. In our experiment, the time cost is only around 1 μs, which is negligible when compared to Put/Get operation. After completing all operations at each module, the server returns the matched results to clients for decryption.

For attribute-based encryption (ABE), we employ the algorithm proposed in [21] to test its performance. The JPBC with

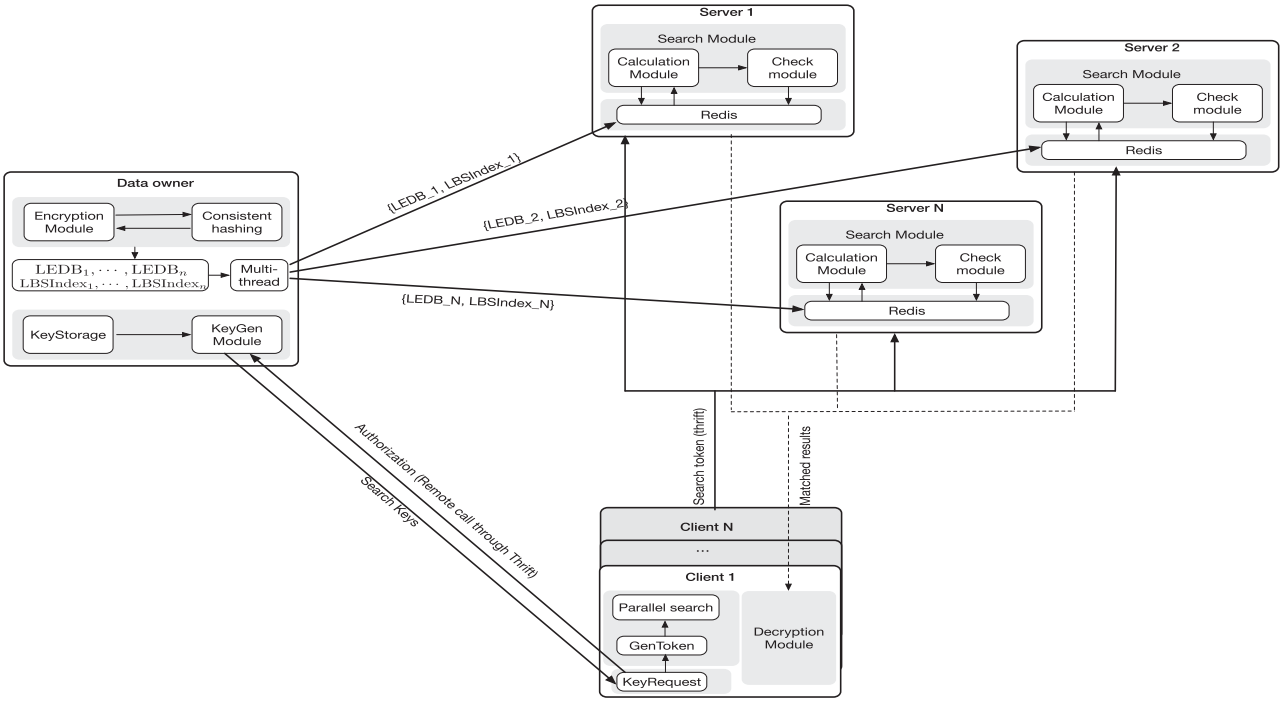

Fig. 6. The system implementation.

the Type A elliptic curve is used to implemented this algorithm. Since ABE is not the main contribution of this paper, we just show its performance with different sizes of attributes, separately with our proposed search solution. Table 1 shows the performance of ABE algorithm when varying the numbers of attributes from 4, 8, to 16. The algorithm runs 10 times for each size and the averaged running times for setup, key generation, encryption, and decryption are shown in the second, third, fourth and fifth columns, respectively.

7.2 Implementation Environment

The goal of this implementation is to evaluate the efficiency of our protocol in terms of data outsourcing time and multi-client Boolean query time. We implement the system prototype and deploy it to the Amazon Web Services (AWS) for experiments. 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), 16 GB RAM and 40 GB SSD. The bandwidth between any two instances is 1Gbps. To achieve remote data operations among different instances, 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 experiment, we choose TPC-H/dbgen [24] to generate the required dataset, which includes a total of 1×10^7 records to represent document IDs. Different number of documents IDs within this dataset will be used in the respective performance studies. The keywords for each document varies from 100 to 10000, and will be specified in the respective performance evaluation.

TABLE 1
The Running Time of ABE Algorithm (ms)

Attributes number	Setup time	KeyGen time	Encrypt time	Decrypt time
4	34.1	58.2	127.1	54.3
8	40.3	98.6	243.3	101.3
16	52.3	178.2	490.2	197.1

7.3 Date Outsourcing Time

This time includes both the data processing time and uploading time. The processing time refers to the data encryption time (building LEDB and LBSIndex) while uploading time refers to the time consumption of transmitting data from data owner to servers.

We first show the impact of server amounts to data outsourcing time. Fig. 7 shows the outsourcing time cost for 1×10^5 document IDs, when we vary the server' amounts

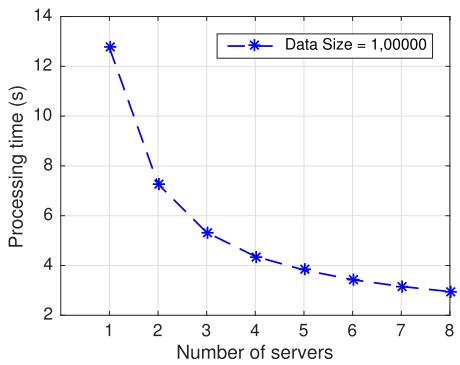

Fig. 7. The processing time with different number of servers.

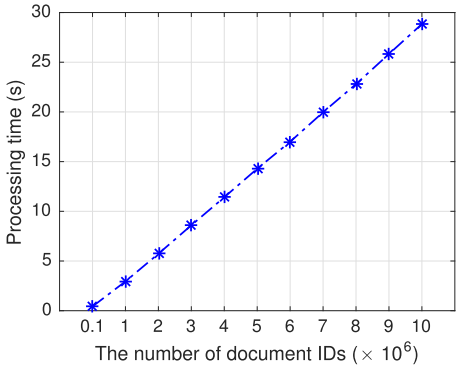

Fig. 8. The processing time with of different sizes of dataset over 8 servers.

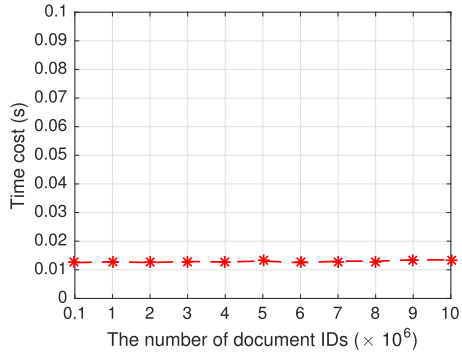

Fig. 10. The time cost with the increasing of documents that contains least frequent keyword.

from 1 to 8. From this figure, we can see that the outsourcing time is substantially decreased with the increasing of server amounts. This is due to the paralleled uploading operations from data owner to multiple servers. In particular, the outsourcing time is only 2.65s with 8 servers comparing to 13.04s with one server. If the server amounts keep increasing, the time costs for outsourcing stage will keep reduced. This demonstrates the advantages of distributed servers over a single centralized server on the data outsourcing operations in term of data encryption and uploading.

Fig. 8 shows the time cost for the outsourcing 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 likely linear with the sizes of dataset. Even the dataset size reaches to 10×10^6, the outsourcing time is less than half minute, i.e., $28.23s$. This shows the outsourcing operation of building local encryption structures LEDB and LBSIndex is very efficient and time cost falls within the acceptable level.

7.4 Query Time Evaluation

To show the efficiency of query performance, we consider that the number of servers increases from 1 to 8, and the size of dataset increases from 0.1×10^6 to 10×10^6. The number of document IDs that contains each keyword will increase with the growth of dataset sizes. Fig. 9 shows the time cost of Boolean search with the increasing number of servers when the dataset sizes are $0.1 \times 10^6, 1 \times 10^6$ and 2×10^6, respectively. 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 server amounts keep 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. The reasons include: 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 as 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 results from the least frequent keyword. We set the number of document IDs that are corresponding to the least frequent keyword as 100. For other queried keywords, each one will be contained in an arbitrary number of documents and this number is at least 100. We consider the number of servers as 8. Fig. 10 shows the query time under 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 frequent keywords are 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 with the dataset sizes.

We then consider the scenarios where the number of documents including the least frequent keyword increases linearly with the sizes of dataset. Fig. 11 shows that the

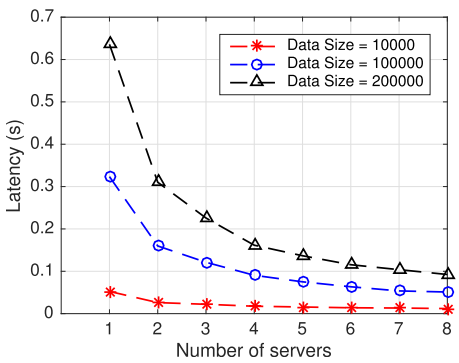

Fig. 9. The search time with different number of servers.

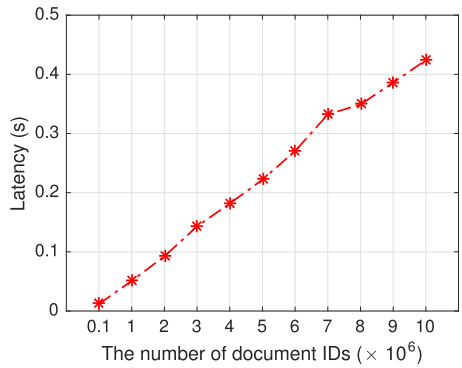

Fig. 11. The time cost with the increasing of document IDs that contains least frequent keyword (linearly increasing with the dataset size).

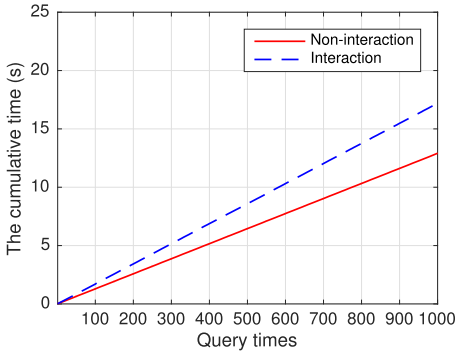

Fig. 12. The comparison of the time under interactive and non-interactive methods with 1000 queries.

query time cost increases almost linearly with the size of the dataset. This represents that the query time increases with the number of documents that contain the least frequent keyword. Therefore, we confirm that the query complexity is decided by the number of documents that contain the least frequent keyword instead of the size of dataset.

7.5 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-interactive scheme, we compare its time consumption with that from the interactive method [5]. 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. 10 and perform 1000 times query. Fig. 12 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 cost is higher when comparing to non-interactive method. The reason is that 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 lots of time and improves the search speed. From Fig. 12, we can see that if a client performs 1000 times query, the total time consumption is 17.20s under the interactive method, while only 12.90s under the non-interactive method. The interactive method will cause 4.3s latency when 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 1 Gbps. 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.

8 Related Works

Searchable Symmetric Encryption. Searchable symmetric encryption [3], [4] 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 [4], [23], [25], [26] 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 [8], [14]. In [8], 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. Their scheme is later improved by Lai et al. [27] with enhanced security and efficiency. After that, Faber et al. extended the above scheme to enable secure range, substring, and wildcard queries [7]. To further improve efficiency, Kamara and Moataz recently proposed a scheme that achieves sublinear search complexity for disjunctive and arbitrary boolean queries [14]. Note that recent SSE schemes also focus on achieving forward and backward privacy in dynamic operations like [28]. Those schemes can readily be integrated to our design.

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 [15], [29] (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 [4], 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 [5]. Here, only authorized keywords can be queried. In addition, they apply the proposed technique to Cash et al.'s scheme [8] for boolean queries. One potential performance issue 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. After that, it 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, thus provide the practitioners little intuition about the integration of production systems and cryptographic primitives. Our design aims to identify and bridge the gaps between them.

Encrypted Data Management Systems. The encrypted database systems [30], [31], [32] have been implemented to support a large portion of database queries over encrypted data. Among them, a system named BlindSeer [30] 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 [11]. 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 [33]. 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.

9 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.

Acknowledgments

This work was supported in part by Louisiana Board of Regents under Contract Numbers LEQSF(2018-21)-RD-A-24, and in part by the Data61-Monash Collaborative Research Project (UbiSENSE for cites), and in part by Research Grants Council of Hong Kong under Grants CityU 11212717 and CityU C1008-16G, and in part by the National Natural Science Foundation of China under Grant number 61572412.

References

[1] "IBM What is big data? Bringing big data to the enterprise." [Online]. Available: www.ibm.com, [Retrieved 2013–08-26].
[2] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Proc. Int. Conf. Theory Appl. of Cryptology Inf. Secur.*, 2010, pp. 577–594.
[3] D.X. Song, D. Wagner, and A. Perrig, "A practcal techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, 2000, Art. no. 44.
[4] R. Curtmola, J. Garay, S. Komara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 79–88.
[5] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur*, 2013, pp. 875–888.
[6] 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. Eur. Symp. Res. Comput. Secur.*, 2016, pp. 154–172.
[7] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2015, pp. 123–145.
[8] 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. Annu. Cryptology Conf.*, 2013, pp. 353–373.
[9] S. Kamara, P. Charalampos, and R. Tom, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur*, 2012, pp. 965–976.
[10] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2013, pp. 258–274.
[11] X. Yuan, X. Wang, C. Wang, C. Chen, and J. Lin, "Building an encrypted, distributed, and searchable key-value store," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 547–558.
[12] R. Escriva, B. Wong, and E. G. Sirer, "HyperDex: A distributed, searchable key-value store," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2012, pp. 25–36.
[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proc. 21st ACM SIGOPS Symp. Operating Syst. Principles*, 2007, pp. 205–220.
[14] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Tech.*, 2017, pp. 94–124.
[15] F. Bao, R. Deng, X. Ding, and Y. Yang, "Private query on encrypted data in multi-user settings," in *Proc. 4th Int. Conf. Inf. Secur. Practice Experience*, 2008, pp. 71–85.
[16] X. Yuan, X. Yuan, B. Li, and C. Wang, "Toward secure and scalable computation in internet of things data applications," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3753–3763, Apr. 2019.
[17] "Redis, an advanced key-value cache and store," [Online]. Available: http://redis.io/, [Online; 2015], 2015.
[18] M. Naveed, S. Kamara, and C.V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 644–655.
[19] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 668–679.
[20] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proc. 29th ACM Symp. Theory Comput.*, vol. 97, 1997, pp. 654–663.
[21] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proc. Int. Workshop Public Key Cryptography*, 2011, pp. 53–70.
[22] Y. Zhang, J. Katz, and C. Papamanthou, " All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. 25th USENIX Conf. Secur. Symp.*, 2016, pp. 707–720.
[23] R. Bost, "Sophos: Forward Secure Searchable Encryption," in *Proc. 23rd ACM Conf. Comput. Commun. Secur.*, 2016, pp. 1143–1154.
[24] Transaction Processing Performance Council, "TPC Benchmark H (Decision Support) Standard Specification," in *TPC*, 2002. [Online]. Available: http://www.tpc.org/information/benchmarks.asp
[25] F. Hahn and F. Kerschbaum, "Searchable Encryption with Secure and Efficient Updates," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 310–320.
[26] 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.
[27] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, D. Mukhopadhyay, R. Steinfeld, S.-F. Sun, D. Liu, and C. Zuo, "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 745–762.
[28] S.-F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 763–780.
[29] C. Dong, C. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," in *Proc. 22nd Annu. IFIP WG 11.3 Working Conf. Data Appl. Secur.*, 2008, pp. 127–143.
[30] V. Pappas, B. Vo, F. Krell, S. Choi, V. Kolesnikov, A. Keromytis, and T. Malkin., "Blind Seer: A scalable private DBMS," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 359–374.

[31] R. Poddar, T. Boelter, and R. A. Popa, "Arx: A strongly encrypted database system," in *Cryptology ePrint Archive, Report 2016/591*, 2016.
[32] R.A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. 23rd ACM Symp. Operating Syst. Principles*, 2011, pp. 85–100.
[33] 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 Asia Conf. Comput. Commun. Secur.*, 2017, pp. 423–435.

Xu Yuan (S'13-M'16) received the BS degree from the Department of Information Security, Nankai University, in 2009, and the PhD degree from the Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, Virginia, in 2016. From 2016 to 2017, he was a post-doctoral fellow of electrical and computer engineering with the University of Toronto, Toronto, ON, Canada. He is currently an assistant professor with the School of Computing and Informatics, University of Louisiana at Lafayette, Louisiana. His research interest include wireless communication, networking, cyber-physical system, security and privacy. He is a member of the IEEE.

Xingliang Yuan (S'15-M'18) received the B.S degree from Nanjing University of Posts and Telecommunications, in 2008, the MS degree from Illinois Institute of Technology, in 2009, both in electrical engineering, and the PhD degree in computer science from City University of Hong Kong, in 2016. He is currently a lecturer with the the Faculty of Information Technology, Monash University, Australia. His research interests include cloud security, privacy-aware computing, and secure networked systems. He is a member of the IEEE.

Yihe Zhang received the BE degree from the School of Information Science and Technology, Sun Yat-sen University, in 2014, and the MS degree from SYSU-CMU Joint Institute of Engineering, Sun-Yat sen University and Carnegie Mellon University, in 2016. He is currently working toward the PhD degree in the School of Computing and Informatics, University of Louisiana at Lafayette. His research interests include machine learning, adversary learning, and social network security. He is a student member of the IEEE.

Baochun Li (F) received the BE degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and the MS and PhD degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000, respectively. Since 2000, he has been with the Department of Electrical and Computer Engineering, University of Toronto, where he is currently a professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include cloud computing, large-scale data processing, computer networking, and distributed systems. In 2000, He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a fellow of the IEEE and a member of the ACM.

Cong Wang (SM'17) received the BE degree in electronic information engineering, and the ME degree in communication and information system, both from Wuhan University, China, and the PhD degree in the electrical and computer engineering from Illinois Institute of Technology. He is currently an associate professor with the Department of Computer Science, City University of Hong Kong. His current research interests include data and computation outsourcing security in the context of cloud computing, blockchain and decentralized application, network security in emerging Internet architecture, multimedia security, and privacy-enhancing technologies in the context of big data and IoT. He is one of the Founding Members of the Young Academy of Sciences of Hong Kong. He received the Outstanding Research Award in 2019, the Outstanding Supervisor Award in 2017, and the President's Awards in 2016 from City University of Hong Kong. He is a co-recipient of the Best Student Paper Award of IEEE ICDCS 2017, the Best Paper Award of IEEE ICPADS 2018, MSN 2015 and CHINACOM 2009. His research has been supported by multiple government research fund agencies, including National Natural Science Foundation of China, Hong Kong Research Grants Council, and Hong Kong Innovation and Technology Commission. He serves/has served as associate editor for the *IEEE Transactions on Dependable and Secure Computing* (TDSC), the *IEEE Internet of Things Journal* (IoT-J) and the *IEEE Networking Letters*, and TPC co-chairs for a number of IEEE conferences/workshops. He is a senior member of the IEEE, and member of the ACM.