Multi-Client Searchable Encryption over Distributed Key-Value Stores

Wanyu Lin, Xu Yuan, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
{*wylin, xuyuan, bli*}*@ece.utoronto.ca*

Cong Wang
Department of Computer Science
City University of Hong Kong
congwang@cityu.edu.hk

***Abstract*—Distributed key-value stores are rapidly evolving to serve the needs of high-performance web services and large-scale cloud computing applications. It is desirable to search directly over an encrypted key-value (KV) store, as data is increasingly stored in the cloud. Encrypted, distributed and searchable key-value stores have been the focus of research, where a data owner outsources his key-value store to a remote server in the cloud in the encrypted form, yet still keeping it searchable. In this paper, we explore the encrypted KV store with the secure multi-client query support. In particular, the data owner can authorize multiple trustable clients (third parties) and allow them to search its encrypted database over KV store. The design goal is to ensure the data confidentiality and query privacy. From the data owner's perspective, the authorized query should not leak too much information thus causing threats to its private database. From clients' perspective, they have the explicit requirement that the query values should not be exposed to the data owner. We design two encryption schemes and token generation methods to satisfy different requirements. To validate the efficiency of our protocols, we implement the system prototype to evaluate their performance.**

I. Introduction

In the age of big data, distributed key-value (KV) store becomes a promising paradigm to serve the needs of large-scale data applications (e.g., medical services, high-performance services, and online gaming, etc.), due to its incremental scalability, exceptional availability, and fault tolerance. Cloud computing technology has been a significant trend in both industry and academia [1]. Its great flexibility and economic savings are motivating the users to outsource their local database management system and intensive computation into the cloud.

However, from the users' perspective, the cloud is intrinsically non-secure. In other words, data outsourcing raises confidentiality and privacy concern. To protect data confidentiality and combat unsolicited accesses in the cloud and beyond, encryption-before-outsourcing technology has been regarded as a fundamental solution [2] [3]. Considering a large amount of data, efficient data utilization after encryption is an especially challenging task in the cloud.

Searchable symmetric encryption (SSE) [4]–[8] is a cryptographic primitive addressing secure and efficient search for outsourcing data. One category of the related research on SSE focuses on practical keyword search (single keyword or boolean keyword). The other group of the related works on SSE focus on supporting secure multi-client search (with client authorization, revocation mechanisms). However, they are not explicit for distributed data stores. Considering the features and requirements in modern KV stores, designing an efficient searchable encrypted key-value store, that supports the multi-client search without privacy breach, remains a challenging open problem.

Very recently, a secure, searchable distributed key-value store (BlindDB) was proposed by Yuan et al. [9]. They built an encrypted local index framework for efficient, simple queries via secondary attributes, while still preserving the functionalities of modern key-value stores. To enable efficient search, they proposed a secure data partition algorithm that can store the encrypted data distributedly across a cluster of nodes. The secure query process can be performed in parallel. Regarding data confidentiality, the leakage to the server is formally specified and proven in [9]. However, the encrypted key-value store only allows data owner itself to perform the secure search via interaction with the remote server.

In this paper, we investigate secure multi-client search over encrypted distributed key-value store. In particular, the data owner can authorize multiple trustable clients (third parties) and allow them to search its encrypted database efficiently via interaction with the server. In our design, we aim to ensure both the data confidentiality and query privacy. Specifically, from the data owner's perspective, the authorized query should not leak too much information so that causing any threats to its private database. From the clients' perspective, the query values should not be exposed to the data owner.

For example, a university outsources an encrypted key-value database which contains students and staffs' personal information (name, sex, address, academic performance, etc.) to the cloud. The client (students, administrators, professors, etc.) can search the store but only via queries authorized by the university according to its policies. Due to the universities' policies, the professor can only search students' scores on his course rather than all of the detailed information about each student. The cloud should also learn as little as possible about the encrypted database and queries submitted by clients. In some cases, the student may not be willing to expose the actual query to the university itself.

For enabling multi-client search over encrypted distributed key-value store in the cloud, we start from the design of BlindDB [9]. Our first contribution lies in extending the BlindDB secure search protocol to support the multi-client

978-1-5090-6517-2/17/$31.00 ©2017 IEEE

setting while preserving its basic query capabilities and performance in [9]. In this extension, the data owner provides the authorized client with a pair of query tokens. Specifically, the server would verify that the data owner has authorized the search tokens submitted by the clients before performing the search protocol. To achieve our objective, we initially use a homomorphic signature mechanism where the search tokens are signed by the data owner and can be verified by the server via a per-query blinding factor. Our primary design preserves the same level of privacy and the same remarkable performance of BlindDB.

Next, to protect clients' privacy, we augment the multi-client setting with blind query support. In this extension, the client queries can be hidden from the data owner rather than being exposed of the query plaintext in our initial design. In this setting, we utilize a particular pseudo-random function (PRF) "oblivious PRF" (OPRF) to generate the search tokens. In other words, by using OPRF, our search tokens can be generated by implementing a two-party computation protocol. In this protocol, the data owner inputs a private key, the client inputs a query, the client can learn the tokens' value, yet the data owner would learn nothing about the query. Thus, the client's query would not be exposed during the query procedure.

The remainder of this paper is organized as follows. In Sec. II, we address our system architecture and threat assumption. In Sec. III, we first recall the basic encrypted distributed key-value store [9] and we explain how to extend support for secure multi-client search. In Sec. IV, we present our real-world implementation on a Redis cluster and evaluate its validity and performance in the geo-distributed servers. We discuss related work and conclude the paper in Sec. V and Sec. VI, respectively.

II. Problem Formulation

A. System Architecture

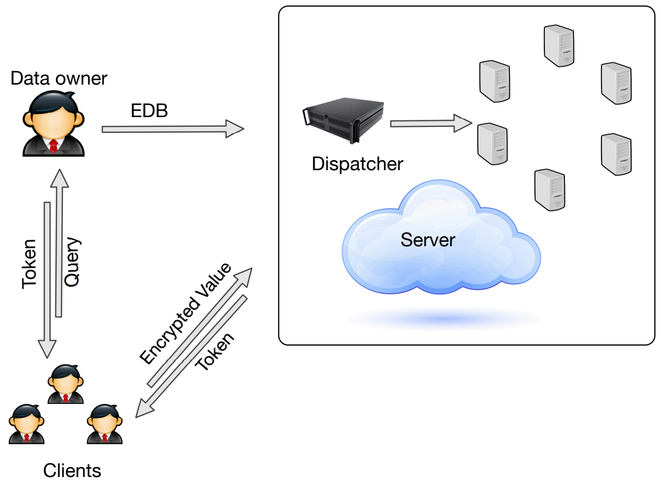

Fig. 1. System architecture of distributed, encrypted key-value store

In this paper, we upgrade the system architecture of distributed, encrypted key-value store proposed in [9] to adapt to a multi-client model as shown in Fig. 1. In this architecture, four entities are included: 1) Data owner D with data application, 2) A dispatcher; 3) Some clustered storage nodes, and 4) Multiple clients. Both dispatcher and storage nodes are deployed in the public cloud or on-premise data centers. The data owner D owns the database. In this system, the data owner performs data encryption, encrypted index construction and then sends its encrypted database **EDB** to the dispatcher. After receiving the encrypted data, the dispatcher can perform the consistent hashing to route them to the target nodes. Finally, the encrypted database would be stored distributedly in the cluster of nodes.

Once there are clients' requests, our multi-client system requires the clients first to obtain search tokens with a signature from the data owner. Subsequently, the client would send its per-query search tokens to the dispatcher. Then the dispatcher in the server module handles search tokens. It first verifies the query tokens with the authentication mechanism. Once the requests are confirmed, the dispatcher would process the authenticated requests over the encrypted indexes and use the APIs of the underlying key-value stores to retrieve the encrypted data records.

Our system aims to ensure efficient and secure query service to the clients via the remote server S. The multi-client setting integrates the secure data partition algorithm so that each storage node can index its local encrypted data and process a given secure query in parallel.

B. Threat Assumption

As shown in Fig. 1, the dispatcher and storage nodes are deployed in the public cloud. For simplicity, we consider them as an entire server component in our threat model.

The server with **EDB** is considered as 'honest-but-curious'. This assumption is consistent with the most related works on searchable encryption [8], [10]. Specifically, it acts in an 'honest' fashion that correctly follows the designated protocol specification. In other words, the server maintains the storage nodes and executes the operations as required. However, it is 'curious' to infer and analyze the encrypted database in its storage. Besides, the server can also monitor the query protocols. It may learn additional information about the data, clients queries and encrypted result records after a number of query protocol executions for an adaptively generated sequential queries.

In our multi-client scenario, we assume that the clients are not in the trusted domain and may collude with each other. Nevertheless, the clients do not collude with the server. It will not expose the data encryption keys to the server. Meanwhile, the server will not respond to the unauthorized requests. The clients may act maliciously, trying to query the information beyond what they are authorized. As the data owner generates the per-query search tokens, he/she may learn additional information about the clients by their requests. Thus, from the perspective of the clients, the data owner may also be a threat. Besides, we also assume that the communication channels are authenticated and encrypted against eavesdropping.

In brief, our goal are threefold in terms of security: 1) leak as little information as possible to *S* about the database and clients' query value; 2) prevent clients from running any other queries than those for which *D* issued them tokens; 3) prevent the data owner learning about the clients' query value.

C. Cryptographic Primitives

A symmetric encryption scheme **SE(KeyGen, Enc, Dec)** consists of three algorithms: the key generation algorithm **KeyGen** takes a security parameter k as an input to return a secret key K_v for data value encryption; The encryption algorithm **Enc** takes a key K_v and a value $v \in \{0,1\}^*$ as input to return a ciphertext $v^* \in \{0,1\}^*$; The decryption algorithm **Dec** takes K_v and v^* as inputs to return v if K_v is exactly the key to encrypt v.

A family of pseudo-random functions (PRF) is defined as: $F : K \times X \rightarrow R$, if for all probabilistic polynomial time, distinguishers Y, $|Pr[Y^{F(k,)} = 1 | k \leftarrow K] - Pr[Y^g = 1 | g \leftarrow \{\mathbf{Func} : X \rightarrow R\}]| < negl(k)$, where $negl(k)$ is a negligible function in k. "Oblivious PRF" [11] is a special case of PRF. A PRF $\mathbf{F}(K, C)$ is called oblivious if there is a two party computation protocol in which the data owner inputs K, the client inputs C, the client learns the value of F but the data owner learns nothing. The high-level description of the two-party computation protocol with oblivious PRF is shown as Fig. 2.

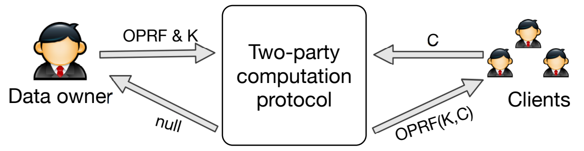

Fig. 2. Two-party computation with oblivious PRF

III. Multi-client Searchable Encryption

This section presents the designs of the multi-client searchable encryption in details underlying the encrypted key-value store proposed in [9].

A. The basic setting of encrypted KV store

We first recall the basic design of the encrypted KV store [9] that forms the basis for our solution to searchable encryption in a more advanced multi-client model. The high-level description of its contribution is that it constructed an encrypted local index framework for efficient queries via secondary attributes, yet still preserving the functionality of the modern key-value stores.

In this system, they first proposed a secure data partition algorithm that dispatches encrypted data records across a cluster of nodes, while preserving horizontal scalability and fault tolerance. Second, they implemented two basic APIs **put** and **get** to support retrieval and update on a single encrypted data record. And then they built an encrypted local index framework towards efficient secure queries via secondary attributes of data in distributed key-value stores. Once the dispatcher receives search tokens, it can perform the consistent hashing to route them to the nodes. Thus, the search protocol can be executed in each node in parallel.

The encrypted local index construction protocol in BlindDB [9] is presented in Fig. 1 ; see [9] for full design rationale and analysis. Here we provide a high-level description as needed for the extension to this protocol we introduce in the following subsections.

Algorithm 1 Build encrypted local indexes

```
Require: Private key: $K$; Row name set: $\mathbf{R}$; Column attribute set: $\mathbf{C}$; Data values: $\mathbf{V}$
Ensure: Encrypted indexes: $\mathbf{I}$.
1: $\mathbf{I} : \{I_1, \cdots, I_n\} \leftarrow init()$;
2: for $i = 1$ to $n$ do
3:     $K_C \leftarrow \mathbf{PRF}(K, i)$;
4: end for
5: for all $C \in \mathbf{C}$ do
6:     init counters $\mathbf{c}$ for $C$: $\{c_1 \leftarrow, \cdots, c_n \leftarrow 1\}$;
7:     for $\forall v \in \mathbf{V}$ associated with $C$ do
8:         $l^* \leftarrow \mathbf{PRF}(K_a, R||C)$, where $R \in \mathbf{R}$;
9:         $i \leftarrow \mathbf{route}(l^*)$;
10:        choose $I_i \in \mathbf{I}, c_i \in \mathbf{c}$ for node $i$;
11:        $t_1 \leftarrow \mathbf{PRF_1}(K_C, 1||C)$, $t_2 \leftarrow \mathbf{PRF_2}(K_C, 2||C)$;
12:        $\alpha \leftarrow \mathbf{G_1}(t_1, c_i)$;
13:        $\beta \leftarrow \mathbf{G_2}(t_2, c_i) \oplus l^*$;
14:        $\mathbf{insert}(\alpha, \beta)$;
15:        $c_i$++;
16:    end for
17: end for
```

The basics of BlindDB secure data partition algorithm is to map key-value pair/record into encrypted one. Specifically, in column-oriented data model, $< l^*, v^* >=< \mathbf{PRF}(K_a, R||C), \mathbf{Enc}(K_v, v) >$, where K_a, K_v are private keys generated by the data owner, R is a row name, C is an attribute name. [9] uses $\mathbf{PRF}(K_a, R||C)$ as the label for partition. The database encryption and encrypted local index construction procedures are executed by the data owner.

Different from [9] which token is generated via PRF on the inputs of the private key and column attribute C, our multi-client setting utilizes an "oblivious PRF" (OPRF) computation between the client ς and the data owner *D* to blind the client's query. Besides, to avoid unauthorized query from the malicious clients, we employ a homomorphic signature mechanism in which the per-query tokens can be signed by the data owner and verified by the server before each search process.

Most changes on BlindDB are in the **GenToken** protocol. The encrypted local indexes construction protocol remains mostly unchanged except for the implementation of the PRF, and **Search** in server side is substantially unmodified except for the additional request's authentication.

B. Multi-Client Setting with Client Authorization

In this subsection, we present an extension of the BlindDB protocol for the multi-client setting in more detail below. Our extension preserves the functionality and performance of the original BlindDB [9], while securely serving multiple clients, all of which can behave maliciously.

Formally, the multi-client setting changes the syntax of the original query scheme in BlindDB. It includes an additional algorithm **GenToken** which on the inputs of the secret key K generated by the data owner D and a query for searching a secondary attribute C's value submitted by the client ς. The **GenToken** protocol is to generate a pair of search-enabling tokens (t_1, t_2). Then the **Search** procedure is executed by the server S on the inputs of the search token (t_1, t_2) and the encrypted database $\mathbf{EDB} = \{\mathbf{I}, <\mathbf{L}^*, \mathbf{V}^*>\}$, where $\mathbf{I}$ is the encrypted indexes, $<\mathbf{L}^*, \mathbf{V}^*>$ is the encrypted key-value records.

For enabling multi-client search over the outsourced key-value stores, an intuitive solution is that the data owner generates the search tokens and then responds to the corresponding client. Then the client with search tokens can perform the search. However, as mentioned in our threat model in Sec. II-B, the clients may collude with each other. In other words, the authorized client may send his/her search tokens to the unauthorized clients. To eliminate this threat, we employ a one-time blinding factor for each query. We describe the changes to the original BlindDB secure query scheme in Fig. 2 needed to support the multi-client setting.

EDBSetup: This pre-processing phase is almost identical to the one in BlindDB. The only difference is that our design requires an additional element K_M to be outsourced in the server for clients authorization. The output from this phase is the private key K kept by D and $\mathbf{EDB} = \{K_M, \mathbf{I}, <\mathbf{L}^*, \mathbf{V}^*>\}$ to be outsourced at the cluster of the storage nodes at the remote side, where K_M is a private key generated by D for clients authorization, $\mathbf{I}$ is the encrypted local indexes generated by executing Algorithm 1, $<\mathbf{L}^*, \mathbf{V}^*>$ are encrypted key-value pairs/records which can be decrypted by K.

GenToken (line $1 - 10$ in Algorithm 2): This is the new multi-client specific phase in which the data owner D, using its private key K_M, authorizes the client ς for a secondary attribute query and provides ς with the necessary tokens to enable search at the server S. As long as the data owner D receives the secondary attribute name submitted by ς, it will perform the following operations. The data owner would generate a one-time blinding factor $\xi \leftarrow Z_p^*$ at the beginning and then set $t_1^* \leftarrow \mathbf{PRF}(K_C, 1||C)^\xi$, $t_2^* \leftarrow \mathbf{PRF}(K_C, 2||C)^\xi$, instead of $t_1 \leftarrow \mathbf{PRF_1}(K_C, 1||C)$, $t_2 \leftarrow \mathbf{PRF_2}(K_C, 2||C)$. Afterwards, the data owner leverages symmetric encryption to encrypt ξ. Thus $\xi^* = \mathbf{AuthEnc}(K_M, \xi)$. Finally, the data owner will send the tokens with the authentication factor to the client. K_v is the private key for search records decryption.

Search (line $11 - 28$ in Algorithm 2): To see how this enables search as in BlindDB, first note that with tokens $(t_1, t_2) = \{\mathbf{PRF_1}(K_C, 1||C), \mathbf{PRF_2}(K_C, 2||C)\}$, the distributed nodes in the server side can access to the encrypted data value in parallel. Before the received tokens are dispatched, the dispatcher would verify the authenticity of the client by decrypting the blinding factor ξ^* and raise the tokens' value to the power of $1/\xi$ to obtain the search tokens. Security relies on the fact that if the client ς provides the server with a value other than the one given by the data owner, then the final tokens will not correspond to the value of the encrypted indices. The remaining **Search** protocol based on tokens (t_1, t_2) on the server side is mostly unmodified except for the additional requests authentication procedure.

Algorithm 2 MC-Secure Query on a given attribute

Require: Queried column attribute: $C \in \mathbf{C}$ from Client; Data owner's private key: K.

Ensure: Given attribute's values: $\mathbf{V_r}$.

```
1:  Data owner:
2:  $\xi \leftarrow Z_p^*$;
3:  for  $i = 1$ to $n$ do
4:      $K_C \leftarrow \mathbf{PRF}(K, i)$;
5:      $t_1^* \leftarrow \mathbf{PRF}(K_C, 1||C)^\xi$, $t_2^* \leftarrow \mathbf{PRF}(K_C, 2||C)^\xi$;
6:  end for
7:  $\xi^* = \mathbf{AuthEnc}(K_M, \xi)$;
8:  Send $\{K_v, (t_1, t_2)_n, \xi^*\}$ to the client.

9:  Client:
10: Send $\{(t_1, t_2)_n, \xi^*\}$ to the server $S$.

11: Search executed in server:
12: Dispatcher:
13: $\xi = \mathbf{AuthDec}(K_M, \xi^*)$;
14: $t_1 \leftarrow t_1^{*1/\xi}$, $t_2 \leftarrow t_2^{*1/\xi}$;
15: $\mathbf{Node}_{1-n}$:
16: for  $i = 1$ to $n$ do
17:     $c_i \leftarrow 1$;
18:     $\alpha \leftarrow \mathbf{G_1}(t_1, c_i)$;
19:     while $\mathbf{find}(\alpha) \neq \perp$ do
20:         $\beta \leftarrow \mathbf{find}(\alpha)$;
21:         $l^* \leftarrow \beta \oplus \mathbf{G_2}(t_2, c_1)$;
22:         $v^* \leftarrow \mathbf{get}(l^*)$;
23:         Add $v^*$ to $\mathbf{V_r}$;
24:         $c_i$++;
25:         $\alpha \leftarrow \mathbf{G_1}(t_1, c_i)$;
26:     end while
27: end for
28: Send $\mathbf{V_r^*}$ to client $\varsigma$.

29: Client:
30: $\mathbf{V_r} \leftarrow \mathbf{Dec}(K_v, \mathbf{V_r^*})$;
```

C. Augmented Multi-Client Setting

In Algorithm 2, the client sends the query to the data owner for token generation. The data owner may learn clients' information by analyzing the queries from the clients. In this subsection, we augment the multi-client setting with blind query support. In this design, the data owner should learn as little as possible about the client queries, while still being able to authorize the client to perform the secure search over the outsourced distributed key-value stores.

Most changes with respect to the multi-client setting are in the **GenToken** protocol from line 1 to 10 in Algorithm 2. **EDBSetup** remains mostly unchanged except for the implementation of the **PRF**s. The implementation of **PRF**s in Algorithm 1 is also replaced by the oblivious PRF. **Search** protocol is unmodified in this augmented protocol.

To blind client query, we use an "oblivious PRF" (OPRF) two-party computation protocol between the client and the data owner, instead of using a regular PRF in line 11 of Algorithm 1 and line 3 of Algorithm 2, respectively. OPRF is a particular case of PRF in which the data owner inputs K and the client inputs C. By applying the two computation protocol with OPRF, the client learns the value of OPRF and the data owner learns nothing. We use the OPRF protocol [7] in which $\mathbf{OPRF}(r, x) = H(C)^r$, where H is a hash function onto G $\{1\}$ where G is a group of prime order p, and r is a random number $\in Zp^*$, x is a value wish to blind.

As in Fig. 3 shown, the client generates a blinding factor r and applies $\mathbf{OPRF}(r, 1||C) = H(1||C)^r$ in line 3, and then sends the blinded query to the data owner. The data owner generates the search token by raising the value to the power of $K_C \times \xi$, where K_C is the private key, and ξ is the blinding factor for authorization. After receiving the search token, the client would raise the tokens' value to the power of $1/r$ as the line 16 shown. The security relies on the fact that the hash function H is a one-way function, the data owner would not know the value of $1||C$ without knowing the random number r.

To decrypt the encrypted value receiving from the server, the data owner also requires sharing the data encryption key K_v to the client. The data owner masks K_v via XORing a_1 as shown in line 12 and sends it along with the generated tokens to the client. Once the client receives the encrypted attribute value from the server, it will unmask K_v via XORing a_1 and execute the decryption procedure to obtain the final results.

Algorithm 3 Augmented MC-Secure Query on a given attribute

Require: A column attribute: $C \in \mathbf{C}$ from Client; Data owner's private key: K.
Ensure: Given attribute's values: $\mathbf{V_r}$.

```
 1: Client:
 2: random $r \in Z_p^*$;
 3: $a_1 \leftarrow \mathbf{OPRF}(r, 1||C)$, $a_2 \leftarrow \mathbf{OPRF}(r, 2||C)$;
 4: Send $(a_1, a_2)$ to data owner $D$.

 5: Data owner:
 6: $\xi \leftarrow Z_p^*$;
 7: for $i = 1$ to $n$ do
 8:     $K_C \leftarrow \mathbf{PRF}(K, i)$;
 9:     $b_1 \leftarrow a_1^{K_C \times \xi}$, $b_2 \leftarrow a_2^{K_C \times \xi}$;
10: end for
11: $\xi^* = \mathbf{AuthEnc}(K_M, \xi)$;
12: $a_1^* \leftarrow a_1 \oplus K_v$;
13: Send $\{(b_1, b_2)_n, \xi^*, a_1^*\}$ to the client.

14: Client:
15: for $i = 1$ to $n$ do
16:     $t_1^* \leftarrow b_1^{1/r}$,, $t_2^* \leftarrow b_2^{1/r}$;
17: end for
18: Send token $\{(t_1^*, t_2^*), \xi^*\}_n$ to the server $S$.

19: Search executed in server:
20: Dispatcher:
21: $\xi = \mathbf{AuthDec}(K_M, \xi^*)$;
22: $t_1 \leftarrow t_1^{*1/\xi}$, $t_2 \leftarrow t_2^{*1/\xi}$;
23: $\mathbf{Node}_{1-n}$:
24: for $i = 1$ to $n$ do
25:     $c_i \leftarrow 1$;
26:     $\alpha \leftarrow \mathbf{G_1}(t_1, c_i)$;
27:     while find$(\alpha) \neq \perp$ do
28:         $\beta \leftarrow$ find$(\alpha)$;
29:         $l^* \leftarrow \beta \oplus \mathbf{G_2}(t_2, c_1)$;
30:         $v^* \leftarrow$ get$(l^*)$;
31:         Add $v^*$ to $\mathbf{V_r}$;
32:         $c_i$++;
33:         $\alpha \leftarrow \mathbf{G_1}(t_1, c_i)$;
34:     end while
35:     Send $\mathbf{V_r^*}$ to client $\varsigma$.
36: end for

37: Client:
38: $K_v \leftarrow a_1^* \oplus a_1$;
39: $\mathbf{V_r} \leftarrow \mathbf{Dec}(K_v, \mathbf{V_r^*})$;
```

IV. Experimental Evaluation

In this section, we present experimental evaluations of our schemes on a large scale dataset with $100,000$ data values in total (10 bytes for each). The results confirm that our extension of the BlindDB protocol can preserve the functionality of [9] and serve multiple clients securely.

A. Prototype Summary

As the prototype implemented in [9], the consistent hash ring was cached at the client for request routing. Thus, there are three components in the system prototype: the data owner, the cluster of storage nodes/server and the client. The data owner generates the encrypted database **EDB** containing encrypted data records, authentication key, and encrypted indices. Meanwhile, the data owner encrypts the clients' request and sends a pair of per-query tokens to the client. The client decrypts the server's responses. The server uses the authentication key to verify and provides search services to

TABLE I
UPLOAD LATENCY WITH DIFFERENT NUMBER OF DATA RECORDS

No. of data records	10,000	50,000	100,000
MC-secure upload latency/s	24.4779	125.4439	215 .8879
Augmented MC-Secure upload latency/s	22.4706	112.2168	218.9765

TABLE II
QUERY LATENCY WITH DIFFERENT NUMBER OF DATA RECORDS

No. of data records	10,000	50,000	100,000
MC-secure query latency/s	1.9908	1.8406	1.8200
Augmented query latency/s	2.0094	1.8260	1.8187

the clients.

The experiments described in the remainder of this subsection were run on the system prototype implemented in [9]. We deployed the system prototype in geo-distributed servers. We create 6 instances as the Redis cluster to store the encrypted indices and records, each of them equipped with 1 vCores, 2GB RAM and 20GB Disk. We build one instance for the data owner and one for the client, respectively. Each one of them equipped with 2 vCores, 4GB RAM, and 40GB Disk. Ubuntu Server 16.04 was installed in each instance.

Likewise, the remote procedure call is implemented via Apache Thrift. The operations on the nodes are implemented via C++. We implemented the cryptographic building blocks using OpenSSL. Besides, we implemented the secure PRF via HMAC-SHA2 and the symmetric encryption via AES/CBC-256. Our multi-client searchable schemes are integrated into the implementation of the distributed encrypted index framework [1] [9].

B. Performance Evaluation

The evaluation on multi-client searchable encryption over distributed key-value stores mainly focuses on the query performance and upload performance.

To show the performance of our multi-client designs, we evaluate the upload latency with different scales of datasets and the query latency for a given attribute for our designs in Sec. III-B and Sec. III-C, respectively. We evaluate the upload latency and the query latency with up to $100,000$ data records. The results are shown in Table I and Table II. The experiments are conducted in the Redis cluster with 6 instances. We conduct the experiments with 50 rounds. From Table I and Table II , we can clearly see that our augmented design can obtain comparable performance while providing query blinding support.

To show that our designs can benefit from the encrypted local index framework, we conduct our experiments using different scales of clusters with 1, 2, 3, 4, 5, 6 storage nodes, respectively. Our experiments use $100,000$ data values and run 50 rounds. Fig .3 and Fig .4 show that both upload and query latency decrease as the number of nodes increases. The reason is that the encrypted local index framework can effectively handle the data uploads and the queries in parallel. Noted that the query latency decreases drastically. With more storage nodes, the queries are performed more effectively. Thus, we can confirm that our extension preserves the functionality and the performance of original distributed key-value stores, while securely serving multiple clients.

[1]An encrypted, distributed, and searchable key-value store: online at https://github.com/CongGroup/BlindDB

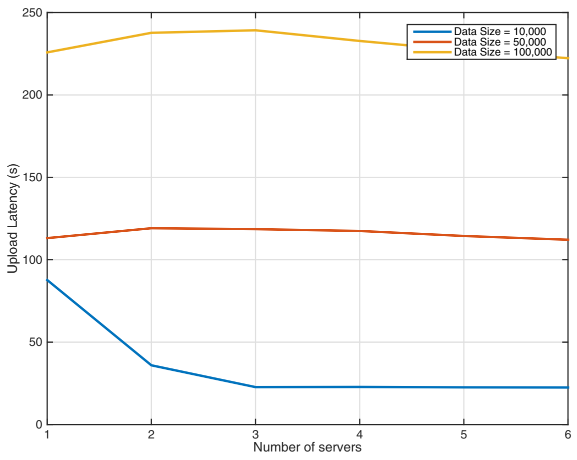

Fig. 3. Augmented MC-secure upload latency with different number of nodes

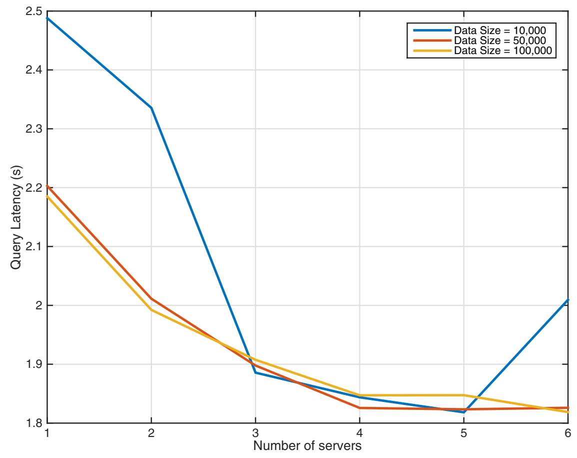

Fig. 4. Augmented MC-secure query latency with different number of nodes

V. RELATED WORK

Searchable Encryption The first searchable encryption scheme is proposed by Song et al. [4]. Goh [5] proposed the first notion of security for searchable encryption. Subsequently, The strong security notion of IND-CKA2 was proposed by Curtmola et al. [6]. In [6], Curtmola et al. proposed a general construction which uses broadcast encryption on top of a single-client scheme to support multi-client secure search. Jarecki et al. proposed another multi-client system called OSPOR-OXT [7] which extended the OXT protocol of Cash et al. [12] to support multi-client setting, while withstanding adversarial non-colluding servers and arbitrarily malicious clients.

Secure Database Systems Security in database systems has been studied in [7], [13]–[16]. These systems can be divided into two categories based on their approaches: software-based systems [7], [13], [14] and hardware-based systems [15], [16]. Among these, CryptDB [13] is a popular software-based system that can execute a wide range of SQL queries on encrypted data. It employs SQL-aware adjustable encryptions with multiple onions to provide strong security strength. Meanwhile, CryptDB requires no changes to the internals of the database management systems. BlindSeer [14] is another practical private database system that can support arbitrary boolean queries, while achieving certain security on a controlled amount of information leakage (e.g., search patterns across multiple queries) [14]. Its scalability is limited by the crucial reliance on bloom filters that requires database sizes whose resultant bloom filters can fit in RAM.

Both Cipherbase [15] and Trusted DB [16] utilize hardware approach to provide data security. In these two systems, cloud server is required to install secure co-processors (SCPUs) where the decryption key is stored on its machine. Attackers could not inspect data stored in an SCPUs memory since SCPUs are tamper-resistant. To answer queries, the server provider sends the encrypted data to the SCPUs for processing and receives encrypted results.

However, above systems are not explicit for distributed data stores. Very recently, an encrypted distributed key-value store is designed and implemented with secure multi-data model support and secure distributed query enabled [9]. In this system, they built an encrypted local index framework to provide secure queries via secondary attributes of data. We make an extension of this system to enable multi-client secure search while preserving functionality and performance of the original system.

VI. Concluding Remarks

In this paper, we focus on the privacy of data owner in multi-client searchable encryption settings over the distributed key-value stores. In some scenarios, the clients may have the explicit requirement that they don't want the data owner to learn any information about the queries. To achieve these objectives, we first employ a homomorphic signature mechanism to authorize the clients, where the search tokens can be signed by the data owner and verified by the server via a per-query blinding factor. Through the one-time blinding factor, we can eliminate the threat result from the replay of the query by different clients. To blind the clients' query, we utilize a two-part computation protocol with obvious PRF to augment our multi-client setting. Last but not the least, for practical usage, we integrate our schemes into an encrypted local index framework so that each node can process the queries in parallel. Our evaluation on the geo-distributed servers demonstrates its efficiency.

References

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[2] S. Kamara and K. Lauter, "Cryptographic Cloud Storage," in *Financial Cryptography and Data Security*, 2010.

[3] C. Wang, K. Ren, and J. Wang, "Secure and Practical Outsourcing of Linear Programming in Cloud Computing," in *Proc. IEEE INFOCOM*, 2011.

[4] D. X. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," in *Proc. IEEE Security and Privacy (S&P)*, 2000, pp. 44–55.

[5] E.-J. Goh *et al.*, "Secure Indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[7] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced Symmetric Private Information Retrieval," in *Proc. ACM SIGSAC conference on Computer & Communications Security*, 2013, pp. 875–888.

[8] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving Multi-keyword Ranked Search over Encrypted Cloud Data," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 25, no. 1, pp. 222–233, 2014.

[9] X. Yuan, X. Wang, J. Lin, C. Wang, and C. Qian, "BlindDB: an Encrypted, Distributed, and Searchable Key-value Store."

[10] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou, "Privacy-preserving Query over Encrypted Graph-structured Data in Cloud Computing," in *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011, pp. 393–402.

[11] M. Naor and O. Reingold, "Number-theoretic Constructions of Efficient Pseudo-random Functions," *Journal of the ACM (JACM)*, vol. 51, no. 2, pp. 231–262, 2004.

[12] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable Searchable Symmetric Encryption with Support for Boolean Queries," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 353–373.

[13] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Processing Queries on an Encrypted Database," *Communications of the ACM*, vol. 55, no. 9, pp. 103–111, 2012.

[14] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin, "Blind seer: A Scalable Private Dbms," in *Proc. IEEE Security and Privacy (S&P)*, 2014, pp. 359–374.

[15] A. Arasu, S. Blanas, K. Eguro, M. Joglekar, R. Kaushik, D. Kossmann, R. Ramamurthy, P. Upadhyaya, and R. Venkatesan, "Secure Database-as-a-service with Cipherbase," in *Proc. ACM SIGMOD International Conference on Management of Data*, 2013, pp. 1033–1036.

[16] S. Bajaj and R. Sion, "Trusteddb: A Trusted Hardware-based Database with Privacy and Data Confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 752–765, 2014.