EncSIM: An Encrypted Similarity Search Service for Distributed High-dimensional Datasets

Xiaoning Liu*, Xingliang Yuan*†, and Cong Wang*†
*City University of Hong Kong, Hong Kong, China
†City University of Hong Kong Shenzhen Research Institute, Shenzhen, 518057, China
xnliu3@cityu.edu.hk, xyuancs@gmail.com, congwang@cityu.edu.hk.

***Abstract*—Similarity-oriented services serve as a foundation in a wide range of data analytic applications such as machine learning, target advertising, and real-time decisions. Both industry and academia strive for efficient and scalable similarity discovery and querying techniques to handle massive, complex data records in the real world. In addition to performance, data security and privacy become an indispensable criterion in the quality of service due to progressively increased data breaches. To address this serious concern, in this paper, we propose and implement "EncSIM", an encrypted and scalable similarity search service. The architecture of EncSIM enables parallel query processing over distributed, encrypted data records. To reduce client overhead, EncSIM resorts to a variant of the state-of-the-art similarity search algorithm, called all-pairs locality-sensitive hashing (LSH). We describe a novel encrypted index construction for EncSIM based on searchable encryption to guarantee the security of service while preserving performance benefits of all-pairs LSH. Moreover, EncSIM supports data record addition with a strong security notion. Intensive evaluations on a cluster of Redis demonstrate low client cost, linear scalability, and satisfied query performance of EncSIM.**

I. Introduction

Similarity-oriented solutions are recognized as one of the key components in advanced data analytics such as machine learning, target advertising, and real-time decisions [1], [2]. To handle a large volume of data records, continuous efforts from industry and academia are made to improve the efficiency and scalability of similarity clustering and querying services [3], [4]. Apart from performance factors, data privacy protection is also considered as a crucial factor regarding the quality of service, because data leaks become a norm nowadays [5]. Data managed in either public storage services or private data centers faces serious threats of being stolen or abused [6].

Applying client-side encryption ensures the end-to-end confidentiality of data records, but it limits the functions for search over the ciphertext. To preserve the functionality, researchers intend to promote solutions in two primary directions. The first is to design cryptographic primitives to enable specific query functions with guaranteed security such as searchable encryption for keyword search [7], [8]. The second direction is to develop encrypted database systems by using suitable cryptographic primitives. Exemplary systems include CryptDB [6], BlindSeer [9], and EncKV [10]. Unfortunately, none of the above solutions supports similarity search.

To bridge the gap, Kuzu *et al*. propose the first practical secure similarity search scheme over encrypted high-dimensional data records [11]. They start from a well-known algorithm for similarity search, aka locality-sensitive hashing (LSH) [1], which returns records within a distance of a given query in sublinear time with constant probability. Then by viewing LSH hashes as keywords, they propose a searchable encryption scheme to enable similarity search in the encrypted domain. Theoretically, any SSE schemes can support similarity search if LSH is applied. However, such treatment only does not deliver a secure, scalable and quality similarity search service for very large datasets. There still exist three challenges.

First, how to design a distributed architecture for similarity search over encrypted high-dimensional data records remains unknown. Such an architecture for large amounts of encrypted data is desired to preserve recognized advantages of plaintext distributed analytics systems such as parallel computing ability and linear scalability. Besides, existing cryptographic primitives do not consider the implementation and deployment in production systems. How to design primitives that can be applied in practice with minimized development efforts are also demanded to be explored.

Second, although LSH boosts query performance compared to brute-force distance computation or traditional spatial indexing techniques [12], it usually demands a large number of hash tables to produce a high-quality result set, e.g., easily reaching a few hundreds in a million-scale dataset [3]. This fact could introduce considerable user overhead when searchable encryption is adopted. As for guaranteed data privacy, the LSH index needs to be generated and encrypted at a trustworthy client before uploading to the server, and hundreds of LSH hashes should also be computed and encrypted at the client for the server to perform secure queries or updates. The above overhead may result in high setup cost and long query latency which are considered as potential bottlenecks to affect service quality especially for large datasets.

Third, one known issue of existing secure similarity search schemes [13], [14] is that adding a new data record would possibly reveal partial information of this record. That is because the encrypted LSH hashes for the server to perform addition can be overlapped with the encrypted hashes of other records if they are queried before. Such leakage allows the server to immediately learn the similarity between the new record and the records already in the dataset. Therefore, the proposed similarity service should enhance the protection of newly added records.

Contributions: We introduce "EncSIM", i.e., a secure and scalable similarity search service for massive encrypted high-dimensional data records. First of all, EncSIM supports distributed similarity search over encrypted data records. We design a modern architecture for EncSIM that enables encrypted data partitioning, parallel encrypted query processing as well as easy deployment of off-the-shelf searchable encryption techniques.

In order to reduce the user overhead, EncSIM resorts to a fast LSH algorithm named "all-pairs LSH" [3], [15], which leverages the idea that LSH functions are reusable. The resulting design reduces the setup time on index building, and greatly decreases encrypted query generation and bandwidth cost. Although the saving is considerable as the number of LSH functions is typically large, directly applying All-pairs LSH may introduce additional leakage from LSH query token reuse (see analysis in Section V-B). To address this issue, we propose a novel technique to randomize the query tokens to achieve comparable security strength of prior schemes. The server only learns the minimized yet necessary information within the query procedure. In the meanwhile, all performance benefits of All-pairs LSH are preserved.

EncSIM also supports secure data addition. To prevent the server from learning the similarity between new records and existing records, we carefully integrate EncSIM's service with the latest searchable encryption technique that achieves forward security [16], and present a secure addition protocol without affecting the correctness and efficiency of the proposed search protocol. As a result, the updates on the encrypted index during data addition will not indicate any useful information about new data records.

The salient features of EncSIM are summarized as follows:

- It enables a secure, fast and scalable similarity search service that can facilitate distributed servers to process encrypted similarity queries in parallel.
- It minimizes the user overhead for outsourced secure similarity search. The setup and query token generation cost is greatly reduced compared to existing designs.
- It provides provable security strength such that the contents of queries and results are strongly protected. The information learned from the service is precisely captured.
- It supports efficient and secure data addition. No partial information of newly added records will be learned.

The prototype of EncSIM is deployed on a Redis cluster. Compared to a basic approach, EncSIM reduces the setup cost by 15%, and brings a saving of $25\times$ on token generation time and a saving of $11\times$ on token bandwidth consumption. The latency of typical queries ranges from 190ms to 500ms, and the throughput reaches 2.6×10^4 of retrieved records per second.

Organization: Section II discusses the related work. Section III describes the preliminaries and notations throughout this paper. After that, Section IV introduces EncSIM's service architecture and threat model in general. We elaborate on EncSIM's design in section V and present the security analysis in Section VI. Finally, we report the implementation and evaluation in section VII and conclude in Section VIII.

II. Related Works

Searchable symmetric encryption: As we aim to enable efficient and scalable services over large encrypted datasets, we are interested in the symmetric key based encrypted search techniques with sublinear time complexity, also known as searchable symmetric encryption (SSE). The first SSE schemes [7], [17] using specific encrypted indexes achieve that the time of keyword search only scales in the number of matched records. Thereafter, secure data dynamic operations are considered in [18]. The recent focus on SSE moves to improve I/O efficiency [19] and support boolean queries [8].

Similarity search on encrypted data: Our design is closely related to secure similarity search schemes on encrypted high-dimensional data records. Under this direction, Lu *et al.* design an encrypted visual vocabulary tree that groups encrypted similar images together [20]. Given a query image, it will be classified to certain visual words. By using order-preserving encryption, encrypted values on each dimension of the query and matched images can be compared to measure the Jaccard similarity. After that, researchers observe that secure similarity search can be solved in a more practical way if LSH is adopted, such that one-way transformed LSH hashes can be used to locate encrypted similar records [11]. In [11], Kuzu *et al.* propose an encrypted bitmap index derived from the LSH inverted index which is the first to enable sublinear secure similarity search. But this scheme is not scalable, because its index size is quadratic in the number of data records, and the data dynamic operations are not supported. Later, Yuan *et al.* [14] devise a high-performance encrypted LSH index to improve the space and query efficiency. Despite improved performance, their design trades accuracy for efficiency and might not be applicable to more general applications. In the meantime, Boldyreva and Chenette formalize the security notion of LSH based searchable encryption schemes and identify open problems on their security [13].

Compared to the schemes above, EncSIM proposed as a comprehensive service has three major contributions. First, it presents a distributed architecture for secure similarity search. This architecture supports encrypted data partition and preserves the capability of incremental scalability and parallel computing in distributed analytic systems. Second, it improves the service quality of secure similarity search services by reducing the setup time and decreasing the query overhead at the user side. Third, it supports secure record addition with strong guarantees.

III. Preliminaries and Notations

A. Preliminaries

Symmetric encryption: A symmetric encryption scheme contains a tuple of three polynomial-time algorithms $(KGen, Enc, Dec)$. The key generation algorithm $KGen$ is a probabilistic algorithm that takes a security parameter λ to return a secret key k; The encryption algorithm Enc is a probabilistic algorithm that takes a key k and a message

$m \in \{0,1\}^*$ to return a ciphertext $c \in \{0,1\}^*$; The decryption algorithm Dec is a deterministic algorithm that takes k and c to return m if c is produced under key k.

Pseudo-random function: Define a family of pseudo-random functions $F : \{0,1\}^\lambda \times \{0,1\}^m \rightarrow \{0,1\}^n$, if for all probabilistic polynomial-time distinguishers $\mathcal{A}$, $|Pr[\mathcal{A}^{F(k,\cdot)} = 1 | k \xleftarrow{\$} \{0,1\}^\lambda] - Pr[\mathcal{A}^g = 1 | g \xleftarrow{\$} \{\mathsf{Func}[m,n]\}]| < negl(\lambda)$, where $negl(\lambda)$ is a negligible function in λ.

Basic LSH: Locality sensitive hashing [1] constructs a randomized data structure to enable approximate but fast similarity search in high-dimensional space. This data structure utilizes a hash function family $\mathcal{H}$ that maps the input data records to a universe U with the property that similar records have hash collisions with a much higher probability than those that are far apart. A general definition of LSH is given below:

***Definition* 1** (Locality-sensitive Hashing)**.** *Let $\mathcal{S}$ be the domain of data records and $\mathcal{D}$ be the distance function. Given distance R_1, R_2, where $R_1 < R_2$, and probability p_1, p_2, where $p_1 > p_2$, a hashing function family $\mathcal{H} = \{h : \mathcal{S} \rightarrow \mathcal{U}\}$ is (R_1, R_2, p_1, p_2)-locality-sensitive if for any $s_i, s_j \in \mathcal{S}$: if $\mathsf{dist}(s_i, s_j) \leq R_1$ then $P[h(s_i) = h(s_j)] \geq p_1$; if $\mathsf{dist}(s_i, s_j) > R_2$ then $P[h(s_i) = h(s_j)] \leq p_2$.*

The basic LSH algorithm concatenates multiple hash functions $h \in \mathcal{H}$ to enlarge the gap between the high probability p_1 and the low probability p_2 [1]. Specifically, one composite LSH function family is defined as $\mathcal{G} = \{g : \mathcal{S} \rightarrow \mathcal{U}^m\}$ such that $g(s) = (h_1(s), \cdots, h_m(s))$, where $h_i \in \mathcal{H}$. To achieve fine accuracy, the algorithm independently and randomly picks L composite LSH functions from $\mathcal{G}$. As a result, each data record is partitioned into L buckets $\{g_1(s), \cdots, g_L(s)\}$.

Given a query record q, the basic LSH algorithm computes $\{g_1(q), \ldots, g_L(q)\}$ and collects candidate data records from pre-built buckets via hash matches. Afterwards, the algorithm evaluates the distance from each candidate record to query q to report the near neighbors within a distance R or rank the candidates if required.

All-pairs LSH: In practice, the number of LSH functions can achieve several hundred [3], [21] for high-quality similarity search services where nearly all the records within the distance radius are returned. To speed up the LSH index construction time and query time, all-pairs LSH known as a fast LSH variant is proposed and implemented in [3], [15]. Basically, it reuses some of the LSH functions to reduce the time of computing LSH hashes. This algorithm generates m partial functions, where each is concatenated by $d/2$ hash functions $h_i \in \mathcal{H}$, i.e., $u(s) = \{h_1(s), \ldots, h_{d/2}(s)\}$, and d is even. Given all $\{u_1, \ldots, u_m\}$, each composite LSH function $g_j \in \{g_1, \cdots, g_L\}$ takes a pair of hashes (u_x, u_y) and concatenates them into a d-dimensional hash, where $1 \leq x < y \leq L$. Consequently, the above combinations produce $L = \binom{m}{2} = m(m-1)/2$ hashes, where $m \approx \sqrt{L}$. Compared to the basic LSH algorithm, all-pairs LSH saves the LSH hash computation time from $O(dL)$ to $O(d\sqrt{L})$.

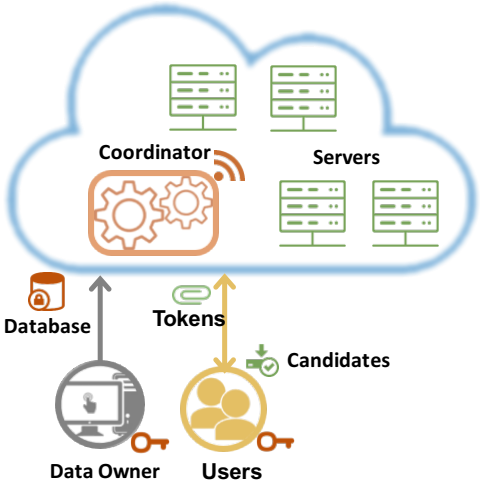


Fig. 1: The service architecture of EncSIM

B. Notations and Functions

This subsection defines the notations used throughout of our paper. We define a hash table H that consists of key-value pairs defined as $\langle l, v \rangle$. Given two binary strings X and Y, $X||Y$ represents their concatenation and $X \oplus Y$ stands for their exclusive disjunction. Our secure similarity search focuses on a high-dimensional data collection. Given a plaintext z-dimensional record s, s^* is the ciphertext of s. Then the data set $\mathbf{S}$ is a record set $\{s_1, \ldots, s_n\}$, and the encrypted dataset $\mathbf{S}^*$ is defined as $\{s_1^*, \ldots, s_n^*\}$. We consider that the identifier id of a record s is also its physical address.

EncSIM's service includes the following primary functions:

- $(\mathcal{I}, \mathbf{S}^*) \leftarrow \mathsf{Setup}(K, \mathbf{S}, N)$: takes as input client's private key K, a high-dimensional dataset $\mathbf{S}$ and the partition number n, and outputs an encrypted index $\mathcal{I}$ in N partitions and an encrypted dataset $\mathbf{S}^*$.
- $\mathbf{t} \leftarrow \mathsf{SearchToken}(K, s_q, \mathbf{lsh})$: takes as input K, client's query record s_q and the parameter set of all-pairs LSH $\mathbf{lsh}$, and outputs a set of search tokens $\mathbf{t}$.
- $\mathbf{R} \leftarrow \mathsf{SimSearch}(\mathbf{t}, \mathcal{I}, \mathbf{S}^*)$: takes as input K, $\mathcal{I}$ and $\mathbf{S}^*$, and outputs a set of similar candidates $\mathbf{R}$.
- $\mathbf{t}_{ad} \leftarrow \mathsf{AddToken}(K, s_{ad}, \mathbf{lsh})$: takes as input K, a new record s_{ad} and $\mathbf{lsh}$, and outputs a set of add tokens $\mathbf{t}_{ad}$.
- $(\mathcal{I}', \mathbf{S}^{*'}) \leftarrow \mathsf{Add}(K, s_{ad}, \mathbf{t}_{ad}, \mathcal{I}, \mathbf{S}^*)$: takes as input K, s_{ad}, $\mathbf{t}_{ad}$, $\mathcal{I}$ and $\mathbf{S}^*$, and outputs the updated index $\mathcal{I}'$ and dataset $\mathbf{S}^{*'}$.

IV. System Model

A. Overview

This section overviews the service architecture of EncSIM and its threat assumptions. Figure 1 illustrates a typical outsourced service scenario that EncSIM targets. A data owner deploys a high-dimensional dataset to the cloud, and delegates the cloud to provide users similarity search services for big data analytics. To ensure the confidentiality of data records, EncSIM runs the Setup function at the data owner side to encrypt the entire dataset before outsourcing. In terms of the

number of servers leased from the cloud, Setup partitions the encrypted data records and builds an encrypted searchable index for each partition (i.e., server).

After the index and data are deployed, the users authorized by the data owner are allowed to use the secure similarity search service provided by EncSIM. To start using the service, the user client runs the SearchToken function to generate encrypted search tokens derived from the LSH hashes of a query record. Then it sends the tokens to a coordinator, which broadcasts the tokens to the servers. After receiving the tokens, each server runs the SimSearch function in parallel. This function uses tokens to process the encrypted index, obtain the ids of candidate records, and return encrypted candidates. At last, the user client decrypts and computes their distances to the query. To add a new record, the data owner encrypts it and runs the AddToken function to generate encrypted add tokens derived from the LSH hashes of this record. Then it sends the tokens and the encrypted file to the target server in terms of the partition. After that, the server runs the Add function to update the encrypted index and dataset.

To speed up encrypted similarity query processing, the architecture of EncSIM follows the existing high performance similarity query engines designed in the plaintext domain [3]. Encrypted records are partitioned by standard distributed data partition algorithms, and the corresponding encrypted indexes are co-located. As a result, all the servers process the query tokens at the same time. Throughout the service, EncSIM provides strong protections on the contents of query and data records. Neither of them is known by the servers.

B. Threat Model

In EncSIM, we consider two types of adversaries. The first is defined as *external* adversaries. They can be outside hackers, who may observe the communication between users and some of the servers, and obtain static snapshots of their encrypted indexes and data records [10], [22]. The second is defined as *internal* adversaries. They can be employees of the cloud provider without authorized access on the contents of data records. In addition to the information that *external* adversaries can learn, they are able to constantly monitor and access the memory and disks of all the servers [10]. Currently, EncSIM does not handle the adversaries who maliciously inject, modify or delete the data records. We note that complementary work (aka. verifiable searchable encryption [23]) would be effective to address the above threat.

V. The design of EncSIM

In this section, we will first introduce a generic service architecture for secure similarity search over distributed and encrypted high-dimensional data records. Based on this service architecture, we introduce a basic construction based on natural integration of all-pairs LSH and searchable encryption. To improve the security strength, we present an optimized construction for EncSIM with preserved performance benefits of all-pairs LSH. Afterwards, we introduce a secure record addition protocol to enhance the service of EncSIM.

Algorithm 1: The $\mathsf{Setup}(K, \mathbf{S}, N)$ function of EncSIM

Input: Data owner's private key: K; Dataset: $\mathbf{S} = \{s_1, \ldots, s_n\}$; Number of servers: N; LSH parameters: $\mathbf{lsh}$.
Output: Encrypted Indexes: $\mathcal{I}$; Encrypted dataset: $\mathbf{S}^*$.
begin
1 //Phase 1: encrypted data partition;
2 **for** $i \leftarrow 1$ *to* n **do**
3 $s_i^* \leftarrow \mathsf{Enc}(K, s_i)$;
4 $j \leftarrow \mathsf{ConsistentHash}(s_i^*)$; // track partition
5 Add s_i to partition $\mathbf{P}_j$;
6 // Phase 2: build an encrypted index per partition;
7 **for** $j \leftarrow 1$ *to* N **do**
8 $I_j \leftarrow \mathsf{BuildIndex}(K, \mathbf{P}_j, \mathbf{lsh})$;

A. Architecture for Secure Distributed Similarity Search

To offer practical secure similarity search services on large encrypted datasets, we envision that EncSIM is desired to inherit advantages of distributed systems such as load balancing, horizontal scalability, and parallel processing [3], [24]. To achieve this goal, our observation is to follow existing distributed architectures and enable EncSIM to support encrypted data partition, and encrypted index and data co-location.

From a high-level point of view, EncSIM employs a standard data partition algorithm, aka consistent hashing, over encrypted data records. Then the cluster of servers is capable to relocate encrypted records without learning the underlying contents of data records, when new servers are added. Note that achieving end-to-end data confidentiality protection requires the data owner to encrypt records and build encrypted indexes for the servers that provide secure similarity search services. To enable encrypted index and data co-location, EncSIM requires the data owner to be aware of the partition of each encrypted record and build the index for each partition respectively. As a result, the servers can later process encrypted queries over their own partition in parallel.

Algorithm 1 presents the setup procedure of EncSIM at the data owner side. There are two phases. The first is to encrypt all the records and perform partition over the ciphertexts. The second phase is to build encrypted similarity index for each partition. The proposed architecture of EncSIM is readily applicable to known practical secure similarity search techniques [11], [14]. Because LSH solves large-scale similarity search through fast hash matches, searchable symmetric encryption (SSE) schemes for encrypted string search can naturally be applied to secure similarity search in the encrypted domain. In EncSIM, we resort to an SSE scheme proposed by Cash *et al.* [19], since the scheme introduces small implementation and deployment efforts. The result encrypted index is stored in a standard hash table which can directly be fit into the production distributed in-memory data store like Redis for high-performance query services.

Algorithm 2: BuildIndex$(K, \mathbf{P}_j, \mathbf{lsh})$

Input: Data owner's private key: K; A partition of data records: $\mathbf{P}$; LSH parameters: $\mathbf{lsh}$.
Output: Encrypted Index: I.
begin
Initialize hash tables $H_{counter}$ and I;
for $\forall s \in \mathbf{P}$ **do**
  for $i \leftarrow 1$ *to* m **do**
    $t_i^1 \leftarrow F_1(K, u_i(s)||i)$, $t_i^2 \leftarrow F_2(K, u_i(s)||i)$;
  for $x \leftarrow 1$ *to* $(m-1)$ **do**
    for $y \leftarrow (x+1)$ *to* m **do**
      // F_1, F_2, and F_3 are secure PRF.
      // Basic construction:
      // $\mathsf{K}_1 \leftarrow t_x^1||t_y^1$, $\mathsf{K}_2 \leftarrow t_x^2||t_y^2$;
      // EncSIM's construction:
      $\mathsf{K}_1 \leftarrow t_x^1 \oplus t_y^1$, $\mathsf{K}_2 \leftarrow t_x^2 \oplus t_y^2$;
      $c \leftarrow H_{counter}.\mathsf{Get}(\mathsf{K}_1)$;
      if $c \neq null$ **then**
        $H_{counter}.\mathsf{Update}(\mathsf{K}_1, c++)$;
      else
        $H_{counter}.\mathsf{Put}(\mathsf{K}_1, 1)$;
      $I.\mathsf{Put}(F_3(\mathsf{K}_1, c), \mathsf{Enc}(\mathsf{K}_2, id))$

Algorithm 3: Basic construction: search protocol

Input: Query: s_q; User's key: K; LSH parameters: $\mathbf{lsh}$; Encrypted indexes: $\mathcal{I}$; Encrypted dataset: $\mathbf{S}^*$.
Output: Candidate set: A.
begin
USER: SearchToken
for $i \leftarrow 1$ *to* m **do**
  $t_i \leftarrow F_1(K, u_i(s_q)||i)||F_2(K, u_i(s_q)||i)$;
Send $\mathbf{t} = \{t_1, \ldots, t_m\}$ to the coordinator;
Coordinator forwards $\mathbf{t}$ to all servers in the cluster.
SERVER: SimSearch
for $\forall t_i \in \mathbf{t}$ **do**
  Parse t_i to $t_i^1||t_i^2$;
for $x \leftarrow 1$ *to* $(m-1)$ **do**
  for $y \leftarrow (x+1)$ *to* m **do**
    $\mathsf{K}_1 \leftarrow t_x^1||t_y^1$, $\mathsf{K}_2 \leftarrow t_x^2||t_y^2$;
    for $c = 1$ *until* $I.\mathsf{Get}(F_3(\mathsf{K}_1, c)) = \perp$ **do**
      $id \leftarrow \mathsf{Dec}(\mathsf{K}_2, I.\mathsf{Get}(F_3(\mathsf{K}_1, c)))$;
      Put s_{id}^* to A;

B. Basic Construction

To better present EncSIM, we first introduce a basic construction that directly combines all-pairs LSH and SSE. In practice, the number of LSH hash functions L is typically large. As indicated in [3], such a parameter can achieve 780 for 1 million high-dimensional tweet vectors, which will bring the considerable cost to the data owner and users. From the perspective of the data owner, each record needs to be applied to all L LSH functions. From the perspective of the user, L tokens need to be generated from a query record for servers to query. To reduce the cost, we propose to leverage all-pairs LSH [3], [21], which reuses some of LSH functions for saving.

Construction: Algorithm 2 presents the index build function for a certain partition of encrypted data records. Given m partial LSH composite functions $\{u_1, \cdots, u_m\}$ defined in Section III-A and each record s in the partition, token t_i^1, t_i^2 are computed via $F_1(K, u_i(s)||i)$, $F_2(K, u_i(s)||i)$, where K is the key, F_1, F_2 are secure PRF. Through $\binom{m}{2}$ combinations of all-pairs LSH, $m(m-1)/2$ tokens are generated, where each token $(\mathsf{K}_1, \mathsf{K}_2)$ is in the form as $(t_x^1||t_y^1, t_x^2||t_y^2)$. Note that the token generation procedure in the basic construction and EncSIM's construction is put together in this algorithm for easy presentation, because this is their only difference in the index build function, which will later be explained in Section V-C. By using the SSE scheme in [19], each token and its matched records are treated as key-value pairs to be encrypted in a standard hash table. In particular, another hash table $H_{counter}$ is used to cache counters to track the number of matched records for each composite LSH function. Accordingly, encrypted key-value pairs are constructed as $(F_3(\mathsf{K}_1, c), \mathsf{Enc}(\mathsf{K}_2, id))$, where F_3 is secure PRF.

In terms of the index build function, the secure similarity search protocol can be initiated as shown in Algorithm 3. The authorized user only needs to compute m tokens $\mathbf{t} = \{t_1, \cdots, t_m\}$ for the coordinator in the cluster. Then the coordinator sends token $\mathbf{t}$ to all the servers for parallel processing. On each server, $\binom{m}{2}$ tokens are generated via combinations, and each token $(\mathsf{K}_1, \mathsf{K}_2)$ is used to access the encrypted index I to find candidate records in sequence via $\mathsf{Dec}(\mathsf{K}_2, I.\mathsf{Get}(F_3(\mathsf{K}_1, c)))$ where c is a self-incremental counter. As the encrypted index and records in one partition are located on the same server, the server can directly return encrypted candidates to the user.

Remark on correctness and performance: The correctness is guaranteed because of the deterministic mapping of the input and output of PRF. The records with matched hashes have corresponding matched tokens just like existing schemes [11], [14]. Regarding performance, all-pairs LSH saves the user computation time and makes the service of EncSIM more user-friendly. As a result, EncSIM benefits from reduced setup time, and a large saving of user computation and bandwidth costs.

Leakage analysis: We observe that such a simple integration of all-pairs LSH and SSE leaks more information beyond the recognized leakage in known SSE schemes. Defined in the security notion of SSE [7], search and access patterns are the only knowledge allowed to be learned. The former pattern tells the repeated tokens appeared in the queries, and the latter pattern tells the accessed index buckets and encrypted records in each query.

Explicitly, if one uses the basic LSH algorithm like the work [11], [14], the server learns the overlapped query tokens between different queries, known as similarity search pattern, defined as $\{u_1(s_{q_i}), \cdots, u_m(s_{q_i})\} \cap \{u_1(s_{q_j}), \cdots, u_m(s_{q_j})\}$ for two queries q_i and q_j. Recall that all-pairs LSH combines two functions into a composite one. Namely, each hash used for lookup is a concatenation of two hashes, i.e., $u_x(s)||u_y(s)$, and its corresponding token is also a concatenation of $t_x||t_y$. Consider that $t_x||t_y$ and $t'_x||t'_y$ are two tokens of two queries, no information should be learned if they are not equal. Unfortunately, this basic approach leaks the information that $t_x = t'_x$ or $t_y = t'_y$ even the above two concatenated tokens are not equal.

C. The Proposed Construction

To improve the security, the challenge is how to minimize the information learned from query tokens generated via all-pairs LSH, while still preserving the correctness of token combinations. Specifically, we devise a search protocol for EncSIM, which can hide the equality of query tokens sent from the user. This protocol includes a novel masking technique to protect query tokens, and a new construction for token combinations such that even permuted tokens are still able to be combined correctly to locate matched candidates.

Design rationale: Before introducing the details, we first explain the high-level idea here. Given a number of tokens generated from a query record, the user utilizes random masks to encrypt the tokens as an additional protection. To allow the server to correctly combine the query tokens, we need to provision the server a capability to cancel the masks without knowing the equality of underlying tokens. As the combined tokens still indicate the similarity search pattern, i.e., the overlap of combined tokens of different queries, we further design a token construction to enable the server to combine the incoming query tokens obliviously. Even if some combined token in two queries is matched, the server will not know the exact mapping of underlying tokens used for combination.

Construction: First of all, the construction of combined token is changed as $t_x^1 \oplus t_y^1$, $t_x^2 \oplus t_y^2$ as shown in line 8 in Algorithm 2. By using the XOR operation instead of concatenation, any combinations of the two tokens will result in the same (K_1, K_2), because XOR is commutative. Such a construction facilitates our following security mechanism that permutes query tokens randomly, which will be introduced later. We also note that XORing two tokens might introduce false positives, but it will not affect the correctness because those false positives can be removed via distance evaluations after retrieval.

The new search protocol of EncSIM is presented in Algorithm 4. Given m query tokens $\{t_1, \cdots, t_m\}$ computed from all-pairs LSH, they are randomly permuted: $\{t'_1, \cdots, t'_m\}$. Then for each token (t'^1_i, t'^2_i), random masks are generated for encryption, i.e., $(t'^1_i \oplus r^1_i, t'^2_i \oplus r^2_i)$. To allow the server to correctly combine the tokens, the masks should be later removed in the search operation. However, directly sending the masks to the server will diminish the protection of query tokens. To solve this problem, we propose to compute $(r^1_i \oplus r^1_{i+1}, r^2_i \oplus r^2_{i+1})$ and send them along with tokens to the server. With $(r^1_i \oplus r^1_{i+1}, r^2_i \oplus r^2_{i+1})$, the server can neither recover t_i nor t_{i+1}. After receiving the tokens, each server parses each token as $t^1_i = (\alpha^1_i, \beta^1_i), t^2_i = (\alpha^2_i, \beta^2_i)$. To combine t_x, t_y where $1 < x < y < m$, the server computes $\mathsf{K}_1 = \alpha^1_x \oplus \alpha^1_y \oplus \beta^1_x \oplus \beta^1_{x+1}, \ldots, \oplus\beta^1_{y-1}, \mathsf{K}_2 = \alpha^2_x \oplus \alpha^2_y \oplus \beta^2_x \oplus \beta^2_{x+1}, \ldots, \oplus\beta^2_{y-1}$. After that, the server uses the combined token to find candidates via $\mathsf{Dec}(\mathsf{K}_2, I.\mathsf{Get}(F_3(\mathsf{K}_1, c)))$ just like the basic construction.

To better illustrate our design, we present an example as follows. For $m = 3$, $\{t_1, t_2, t_3\}$ are firstly generated. After permutation, $\{t_2, t_1, t_3\}$ are derived, which are viewed by the server as $\{t'_1, t'_2, t'_3\}$. Then the user computes $\{t'_1 \oplus r_1, t'_2 \oplus r_2, t'_3 \oplus r_3\}$ and $\{r_1 \oplus r_2, r_2 \oplus r_3\}$ for the servers. At the server side, $t'_1 \oplus t'_2$ is derived from $t'_1 \oplus r_1 \oplus t'_2 \oplus r_2 \oplus r_1 \oplus r_2$, $t'_2 \oplus t'_3$

Algorithm 4: EncSIM: search protocol

Input: Query: s_q; User's key: K; LSH parameters: $\mathbf{lsh}$; Encrypted indexes: $\mathcal{I}$; Encrypted dataset: $\mathbf{S}^*$.
Output: Candidate set: A.
begin

USER: SearchToken
1 $\{r^1_1, \ldots, r^1_m\} \xleftarrow{\$} (0,1)^\lambda$; $\{r^2_1, \ldots, r^2_m\} \xleftarrow{\$} (0,1)^\lambda$;
2 **for** $i \leftarrow 1$ *to* m **do**
3 $t^1_i \leftarrow F_1(K, u_i(q)||i)$, $t^2_i \leftarrow F_2(K, u_i(q)||i)$;
4 $t_i \leftarrow t^1_i||t^2_i$;
5 Randomly permute $\{t_1, \cdots, t_m\}$ to $\{t'_1, \cdots, t'_m\}$;
6 **for** $i \leftarrow 1$ *to* m **do**
7 **if** $i < m$ **then**
8 $t'^1_i \leftarrow (t'^1_i \oplus r^1_i, r^1_i \oplus r^1_{i+1})$;
9 $t'^2_i \leftarrow (t'^2_i \oplus r^2_i, r^2_i \oplus r^2_{i+1})$;
10 **else**
11 $t'^1_i \leftarrow t'^1_i \oplus r^1_i$;
12 $t'^2_i \leftarrow t'^2_i \oplus r^2_i$;
13 Send $\mathbf{t} = \{t'_1, \ldots, t'_m\}$ to the coordinator;
14 Coordinator forwards $\mathbf{t}$ to all servers in the cluster.

SERVER: SimSearch
15 **for** $i \leftarrow 1$ *to* m **do**
16 parse t_i to $t^1_i||t^2_i$;
17 **if** $i < m$ **then**
18 $(\alpha^1_i, \beta^1_i) \leftarrow t^1_i$, $(\alpha^2_i, \beta^2_i) \leftarrow t^2_i$;
19 **else**
$\alpha^1_i \leftarrow t^1_i$, $\alpha^2_i \leftarrow t^2_i$;
20 **for** $x \leftarrow 1$ *to* $(m-1)$ **do**
21 **for** $y \leftarrow (x+1)$ *to* m **do**
22 $\mathsf{K}_1 \leftarrow \alpha^1_x \oplus \alpha^1_y \oplus \beta^1_x \oplus \beta^1_{x+1}, \ldots, \oplus\beta^1_{y-1}$;
23 $\mathsf{K}_2 \leftarrow \alpha^2_x \oplus \alpha^2_y \oplus \beta^2_x \oplus \beta^2_{x+1}, \ldots, \oplus\beta^2_{y-1}$;
24 **for** $c = 1$ *until* $I.\mathsf{Get}(F_3(\mathsf{K}_1, c)) = \perp$ **do**
25 $id \leftarrow \mathsf{Dec}(\mathsf{K}_2, I.\mathsf{Get}(F_3(\mathsf{K}_1, c)))$;
26 Put s^*_{id} to A;

Algorithm 5: EncSIM: record addition protocol

Input: Data owner's private key: K; New record: s; LSH parameters: $\mathbf{lsh}$; State table: $\mathbf{st}$, $\mathbf{ST}$.
Output: Updated Index: I.
begin

DATA OWNER:

$s^* \leftarrow \mathsf{Enc}(K, s)$;
$j \leftarrow \mathsf{ConsistentHash}(s^*)$; // track partition
for $i \leftarrow 1$ *to* m **do**
  $t_i^1 \leftarrow F_1(K, u_i(s)||i)$, $t_i^2 \leftarrow F_2(K, u_i(s)||i)$;
  if $\mathbf{st}.\mathsf{Get}(t_i^1) = \perp$ **then**
    $st_i \xleftarrow{\$} \{0,1\}^\lambda$;
for $x \leftarrow 1$ *to* $(m-1)$ **do**
  for $y \leftarrow (x+1)$ *to* m **do**
    $\mathsf{K}_1 \leftarrow t_x^1 \oplus t_y^1$, $\mathsf{K}_2 \leftarrow t_x^2 \oplus t_y^2$;
    $ST_{sc}||sc \leftarrow \mathbf{ST}.\mathsf{Get}(\mathsf{K}_1)$;
    if $ST_{sc}||sc = \perp$ **then**
      $st_x \leftarrow \mathbf{st}.\mathsf{Get}(t_x^1)$, $st_y \leftarrow \mathbf{st}.\mathsf{Get}(t_y^1)$;
      $ST_0 \leftarrow st_x \oplus st_y$, $sc \leftarrow -1$;
    else
      $ST_{sc+1} \leftarrow \pi_{sk}^{-1}(ST_{sc})$;
    $\mathbf{ST}.\mathsf{Put}(\mathsf{K}_1, ST_{sc+1}||sc+1)$;
    $UT_{sc+1} \leftarrow F_3(\mathsf{K}_1, ST_{sc+1})$;
    Add $(UT_{sc+1}, \mathsf{Enc}(F_3(\mathsf{K}_2, ST_{sc+1}), id)$ to A;
Send A, s^* to server j.

SERVER j:

Put all key-value pairs in A to I;

is derived from $t_2' \oplus r_2 \oplus t_3' \oplus r_3 \oplus r_2 \oplus r_3$, and $t_1' \oplus t_3'$ is derived from $t_1' \oplus r_1 \oplus t_3' \oplus r_3 \oplus r_1 \oplus r_2 \oplus r_2 \oplus r_3$. In essence, the server obtains $t_2 \oplus t_1$, $t_1 \oplus t_3$, and $t_2 \oplus t_3$, but it will not learn the underlying combination.

Security guarantee: The above search protocol guarantees the minimized information public to external adversaries. The masked and permuted tokens prevent them from learning the search pattern. As the indexes and data records are encrypted, even those adversaries obtain their copies occasionally, they never learn any useful information about the queries, indexes, and data records. For internal adversaries, EncSIM indicates the comparable information compared to prior work [11], [14]. Since they might be able to monitor the query procedure, they would know the similarity search pattern, i.e., the overlapped combined tokens across different queries, and the access pattern, i.e., the candidates of each query.

D. Secure Data Addition

EncSIM also supports data addition when the data owner has new encrypted data records to be included in the service. However, most of the prior dynamic SSE schemes [14], [18] would reveal partial information about a newly added record, i.e., its tokens appeared in previous queries. In the context of similarity search, without search, the server could learn the similarity between the new record and some old records. To solve this issue, EncSIM adopts the latest technique achieving forward security [16], such that the above information is protected in record addition. The idea is to use one-way permutation to make tokens stateful. The tokens of new records are generated via new states.

To preserve the benefit of all-pairs LSH, we carefully integrate the above technique into the construction of EncSIM. The detailed protocol of secure data addition is introduced in Algorithm 5. Given a new record s, the data owner first encrypts it to get s^*, and computes consistent hashing to locate its partition. Next, m tokens are computed from m partial composite hashes. For each token t_i, if the corresponding state is not found in a table $\mathbf{st}$, a new state st_i is randomly generated. Then for each combined token $t_x \oplus t_y$, its initial state ST_0 is derived as $st_x \oplus st_y$, and its subsequent state is derived from a one-way permutation π_{sk}^{-1} of the previous state ST_{sc}, where ST_{sc} is cached in another table $\mathbf{ST}$. Afterwards, each key-value pair is generated via a stateful count: $(F_3(\mathsf{K}_1, ST_{sc+1}), \mathsf{Enc}(F_3(\mathsf{K}_2, ST_{sc+1}), id)$.

To perform search, the user now needs to send the states of query tokens $\{st_1, \cdots, st_m\}$ to the server. After token combinations, the server finds the last state of each combined token $ST_c||c$ in $\mathbf{ST}$, and starts to find the candidates one by one via $\mathsf{Dec}(F_3(\mathsf{K}_2, ST_c), I.\mathsf{Get}(F_3(\mathsf{K}_1, ST_c))$. The subsequent state ST_{c-1} can be derived through $\pi_{pk}(ST_c)$, where (pk, sk) is the key-pair of one-way permutation π. We refer the reader to [16] for its underlying construction based on RSA.

VI. SECURITY ANALYSIS

This section quantifies the security strength of EncSIM. The methodology is to use the security notion of searchable symmetric encryption [7], [18] to prove that EncSIM protects the contents of queries, indexes and data records throughout the service, and only indicates controlled auxiliary information.

Basically, we first define the leakage in the setup. The leakage function $\mathcal{L}_1$ for the Setup function is given as follows:

$$\mathcal{L}_1(\mathbf{S}) = (N, \{n_1, \cdots, n_N\}, |s^*|, |l|, |v|)$$

where N is the number of partitions, $\{n_1, \cdots, n_N\}$ are the number of encrypted records in each partition, $|s^*|$ is the size of ciphertext, $|l|, |v|$ are the size of key-value pairs in the encrypted indexes. We note that the above is the only information to be learned if the adversary can only obtain the images of the index and database.

Then, we define the leakage in the search protocol, i.e., similarity search pattern $\mathcal{L}_2$ and access pattern $\mathcal{L}_3$. $\mathcal{L}_2$ is defined as follows:

$$\mathcal{L}_2(s_q) = (\{|\{g_1(s_q), \cdots, g_L(s_q)\} \cap \{g_1(s_i), \cdots, g_L(s_i)\}|\}_q)$$

where $i \in [1, q]$). This function outputs that for a given query s_q, the server can learn the size of overlapped combined tokens between s_q and previous queries. Recall that our proposed search protocol protects the tokens before combinations, and

thus the security is comparable to existing secure similarity search schemes based on SSE [11], [14]. The access pattern $\mathcal{L}_3$ is defined as follows:

$$\mathcal{L}_3(s_q) = (\{\{\langle l, v\rangle\}_{M_i}, \{s^*\}_{M_i}\}_N, i \in [1, N])$$

This function shows that for each partition, the server knows the accessed key-value pairs in the index, and the matched encrypted records. Based on the above functions, we define the security notion of EncSIM as follows:

***Definition* 2.** *Let* $\Phi = (\mathsf{Setup}, \mathsf{SearchToken}, \mathsf{SimSearch})$ *be the functions in EncSIM. Given leakage functions* $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$*, a probabilistic polynomial time adversary* $\mathcal{A}$ *and simulator* $\mathcal{S}$*, we define two games* $\mathbf{Real}^{\Phi}_{\mathcal{A}}(\lambda)$ *and* $\mathbf{Ideal}^{\Phi}_{\mathcal{A},\mathcal{S}}(\lambda)$ *below:*

$\mathbf{Real}^{\Phi}_{\mathcal{A}}(\lambda)$*:* $\mathcal{A}$ *choose a dataset* $\mathbf{S}$*.* $\mathcal{A}$ *challenger generates* $K \xleftarrow{\$} \{0,1\}^{\lambda}$*, runs* Setup*, and outputs* $\mathcal{I}, \mathbf{S}^*$ *to* $\mathcal{A}$*. Then* $\mathcal{A}$ *adaptively sends* q *query records to proceed the search protocol. For each query, the challenger generates token* $\mathbf{t}$ *from* SearchToken *for* $\mathcal{A}$ *to query the server through* SimSearch*. In the end,* $\mathcal{A}$ *returns a bit as the output.*

$\mathbf{Ideal}^{\Phi}_{\mathcal{A},\mathcal{S}}(\lambda)$*:* $\mathcal{A}$ *choose a dataset* $\mathbf{S}$*.* $\mathcal{S}$ *outputs simulated* $\widetilde{\mathcal{I}}, \widetilde{\mathbf{S}^*}$ *based on* $\mathcal{L}_1$*. For* q *adaptive queries,* $\mathcal{S}$ *outputs simulated* $\widetilde{\mathbf{t}}$ *based on* $\mathcal{L}_2$ *and* $\mathcal{L}_3$ *for* $\mathcal{A}$ *to query the server. In the end,* $\mathcal{A}$ *returns a bit as the output.*

EncSIM is $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$*-secure against adaptive chosen-keyword attack if for all probabilistic polynomial time adversaries* $\mathcal{A}$*, there exists a simulator* $\mathcal{S}$ *such that*

$$Pr[\mathbf{Real}^{\Phi}_{\mathcal{A}}(\lambda)] - Pr[\mathbf{Ideal}^{\Phi}_{\mathcal{A},\mathcal{S}}(\lambda)] \leq negl(\lambda).$$

where $negl$ *is a negligible function in* λ*.*

Accordingly, we give the following theorem:

***Theorem* 1.** *EncSIM is* $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$*-secure against adaptive chosen-keyword attacks in the random oracle model if* F_1, F_2, F_3 *are secure PRF and* $(\mathsf{Enc}, \mathsf{Dec})$ *are semantically secure.*

Proof. We will prove that the adversary $\mathcal{A}$ can hardly differentiate the views of interacting with the real server from the views of interacting with the simulator $\mathcal{S}$. In terms of $\mathcal{L}_1$, $\mathcal{S}$ knows the number of partitions N and the number of data records in each partition $\{n_1, \cdots, n_N\}$. Thus, $\mathcal{S}$ generates a random string $|\widetilde{s^*}|$ with the size of $|s^*|$ to simulate each encrypted record. For partition i from 1 to N, $\mathcal{S}$ further generate Ln_i dummy key-value pairs with the size of $|l|, |v|$ to simulate the index $\widetilde{I}_i$, where L is the number of LSH composite functions. As a result, the real and simulated indexes, and the real and simulated encrypted records are computationally indistinguishable.

To simulate the first query, $\mathcal{S}$ simulates the token set $\widetilde{\mathbf{t}} = \{\widetilde{t_1}, \cdots, \widetilde{t_m}\}$ with random strings, and it then obtains $L{=}\binom{m}{2}$ tokens $\{\widetilde{\mathsf{K}_1}, \widetilde{\mathsf{K}_2}\}$. For partition i from 1 to N, $\mathcal{S}$ uses random oracle H to find key-value pairs, i.e., $\widetilde{I}_i.\mathsf{Get}(H(\mathsf{K}_1, c))$ for all tokens, where c increments from 0, and the sum of all counters $\{c\}_L$ is equal to M_i. By replacing $\mathsf{Enc}(\widetilde{\mathsf{K}_2}, id)$ with

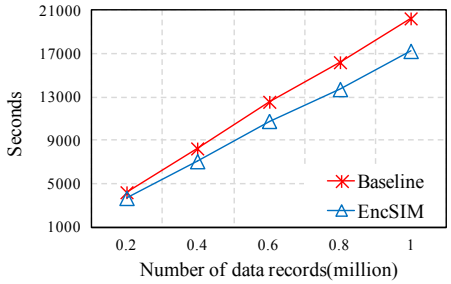


Fig. 2: Setup time

$H(\widetilde{\mathsf{K}_2}||r) \oplus id$, id can be obtained which is the same as shown $\mathcal{L}_3$. Based on $\{id\}_{M_i}$, $\mathcal{S}$ gives the simulated records $\{\widetilde{s^*}\}_{M_i}$. For the subsequent query, $\mathcal{S}$ learns the number of overlapped combination tokens from $\mathcal{L}_2$, denoted as b'. Thus, it also learns the number of matched ones in m tokens, denoted as b, where $b'{=}\binom{b}{2}$. After that, $\mathcal{S}$ selects b simulated tokens from the first query which leads to b' overlapped combination tokens, and generates random tokens for the rest of $m-b$ tokens. For the combination tokens appeared before, $\mathcal{S}$ copies previous results. Otherwise, it follows the same way of simulating results in the first query. Due to the pseudo-randomness of F_1, F_2, F_3, the simulated and real tokens are indistinguishable. The size of overlapped tokens among any two queries is also consistent in two games. Finally, the result candidate ids are identical. Therefore, $\mathbf{Ideal}^{\Phi}_{\mathcal{A},\mathcal{S}}(\lambda)$ are indistinguishable. □

VII. Experimental Evaluation

A. Setup

To evaluate the performance of EncSIM, we implement a prototype via ∼5000 lines of Java code, and publish the code at Github[1]. We deploy this prototype on a cluster of Redis(v3.2.8) servers[2], which consist of 1 job submission node and 5 execution nodes with encrypted indexes and corresponding encrypted records stored. Each node has two Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz, 256 GB RAM and CentOS 6.4 installed. EncSIM uses the cryptographic package of JRE 1.7 for cryptographic primitives. Secure PRF is implemented via HMAC-SHA1. The symmetric encryption is constructed as $(r, \mathsf{HMAC\text{-}SHA1}(K, r) \oplus record)$, where r is a nonce. The measurements reported in this section make use of a database derived from a twitter collection named Sentiment140[3], which contains over millions of tweets. For each tweet, we encode a sparse binary vector in a 374,747-dimension space based on the size of vocabulary extracted. Here, we apply Hamming distance as the distance measure, and the all-pairs LSH parameters are selected as $d = 16, m = 40, L = 780, radius = 15$.

B. Evaluation

We report the efficiency and scalability of EncSIM via a set of performance factors, i.e., setup time, token generation time, token bandwidth cost, query time, throughput, record

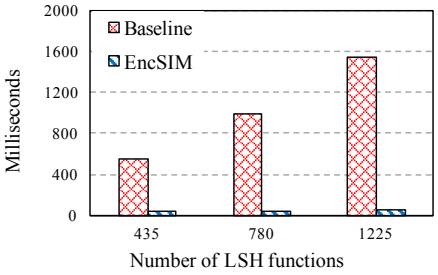
[1]EncSIM prototype: https://github.com/CongGroup/IWQOS-2017-EncSIM

[2]High Throughout Computing Cluster at City University of Hong Kong: http://cslab.cs.cityu.edu.hk/services/high-throughput-computing-cluster-htcc.

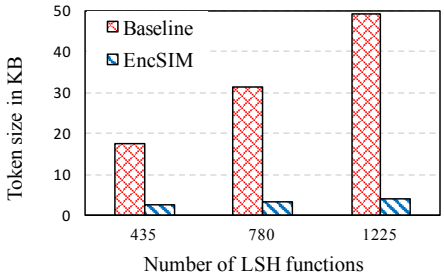
[3]Sentiment140 Twitter Data: http://help.sentiment140.com/for-students/

Schemes	Token computation	Token size
Baseline	$O(dL)$	$O(dL)$
EncSIM	$O(d\sqrt{L})$	$O(d\sqrt{L})$

TABLE I: Complexity comparison: the complexity of baseline is applied to the direct usage of LSH and any SSE schemes.



(a) Token generation time



(b) Token bandwidth consumption

Fig. 3: User side evaluation

addition time, and accuracy. For comparison, we implement a construction denoted as "baseline" that directly combines the basic LSH algorithm and the SSE scheme in [19]. We also implement the basic LSH algorithm denoted as "plaintext" to evaluate the security overhead.

Figure 2 displays the index building time for EncSIM and baseline. Both of them increase linearly according to the number of data records, while EncSIM shortens around 15% time consumption compared with baseline. For an encrypted index with 1 million data records, there is overall 50 minutes saved by our design. The reduction stems from the less computation cost of all-pairs LSH in EncSIM. This advantage is more obviously reflected in secure token generation as Figure 3-(a) illustrated. We choose three groups of LSH functions: 435, 780, and 1225 numbers of composite LSH functions and evaluate the token generation time in milliseconds. As shown in Table I, the time of baseline ascends sharply followed by the growth of L, i.e., the number of LSH functions. On contrast, such cost of EncSIM barely goes up as a result of the token computation complexity is $O(\sqrt{L})$. Although our design involves in the XOR operations, it takes negligible time compared with LSH and PRF calculations. Similarly, Figure 3-(b) also provides the empirically observable evidence of bandwidth retrenchment contributed by all-pairs LSH. The growing number of LSH functions are attributed to the rise of bandwidth overhead for baseline ($L|\mathsf{HMAC\text{-}SHA1}|$), yet EncSIM remains a flat trend ($m|\mathsf{HMAC\text{-}SHA1}|+(m-1)|r|$),

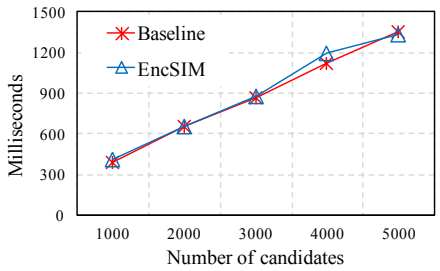


Fig. 4: Server search time

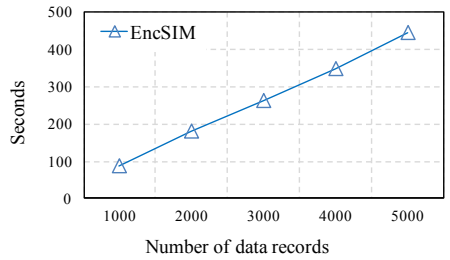


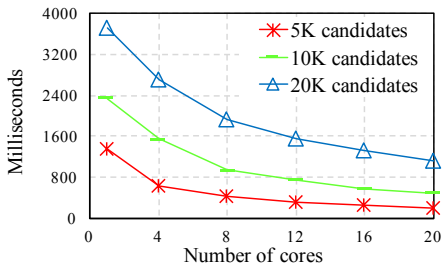
Fig. 5: New record addition time

where $|\mathsf{HMAC\text{-}SHA1}|$ is 20-byte and the randomness r is also 20-byte. Overall, for $L = 1225$, the token generation time brings a saving of 25×, and the token size shrinks over 11 × compared to baseline.

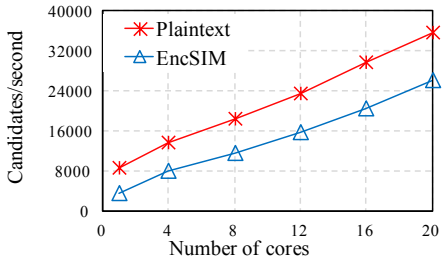
Then we turn our attention from the user side workload to server runtime evaluation. The search time as well as the insertion time present in chunks of 1K data records in the single node mode. According to Figure 4 providing, both schemes formed a similarly uplifted pattern during the gradual rise of returned results. Such outcome indicates that EncSIM does not aggravate overhead on the server side compared with baseline. The record addition of EncSIM includes tweet vector generation, encrypted key-value pairs computation, and put operations to the Redis. As exhibited in Figure 5, the insertion time ascends linearly based on the amount of new records.

The scalability of EncSIM is confirmed through search time and throughput. Figure 6 demonstrates how the performance of EncSIM improves with increasing the number of cores. We can observe a dramatic speedup in similar proportions of queries that return a specified number of candidates in Figure 6-(a). The time consumption with 20 cores is approximately a quarter of it with a single core. Queries that return 5K and 10K take around 190ms and 500ms. Likewise, as Figure 6-(b) displays, the throughput of EncSIM performs sustainably climbing accompany with the incremental cores. In particular, the peak number of candidates processed with 20 cores represents reaches 2.6×10^4 per second, a loss of 26% to the plaintext.

The accuracy is measured on running random 1000 queries selected from the dataset. EncSIM reaches 96% recall rate for range queries. We also report the precision of Top-k queries. The precision is qualitative via the definition ($\frac{1}{k}\sum_{i=1}^{k}\frac{dist(s_i'-s_q)}{dist(s_i-s_q)}$), where s_q is the query record, s_i is the i-th nearest neighbor to s_q of LSH, and s_i' is the ground



(a) Search time



(b) Throughput

Fig. 6: Scalability evaluation

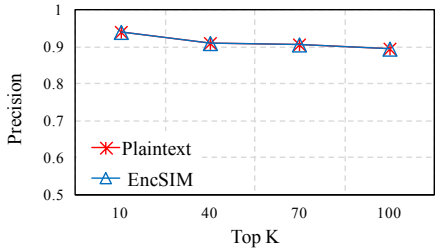


Fig. 7: Accuracy evaluation

truth i-th nearest neighbor. As Figure 7 demonstrates, although the precisions appear a degradation along with the raise of k with tiny fluctuation, all of them remain 0.9. Note that our security design does not affect the correctness of LSH due to the deterministic property of PRF.

VIII. Conclusion

In this paper, we design and develop a secure and scalable service named EncSIM for similarity search over distributed, encrypted data records. EncSIM partitions the encrypted records across a cluster of servers and allows the encrypted index to be stored with the records together, so that EncSIM enables all the servers to process the secure similarity query in parallel. To reduce the user overhead, we adopt a variant of LSH algorithms called all-pairs LSH, and carefully design a query protocol for EncSIM to minimize the leakage while preserving the performance benefits. Besides, EncSIM provides a secure data addition protocol that achieves forward security. Intensive performance evaluations on a Redis cluster demonstrate the efficiency and scalability of EncSIM.

Acknowledgment

This work was supported in part by Research Grants Council of HK (Project No. CityU 11276816 and CityU C1008-16G), CityU of HK (Project No. CityU 9680157 and CityU 9678126), the Natural Science Foundation of China under Project 61572412, and Innovation and Technology Commission of Hong Kong under ITF Project ITS/307/15.

References

[1] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Communications of the ACM*, vol. 51, pp. 117–122, 2008.

[2] simMachine, "simMachine: similarity search & pattern recognition." Online at http://simmachines.com, 2017.

[3] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey, "Streaming similarity search over one billion tweets using parallel locality-sensitive hashing," *Proc. VLDB Endow.*, vol. 6, no. 14, pp. 1930–1941, 2013.

[4] A. Rodriguez and K. Chu, "Locality sensitive hashing by Spark." Online at https://spark-summit.org/2016/events/locality-sensitive-hashing-by-spark/, 2016.

[5] InformationIsBeautiful.net, "World's biggest data breaches." Online at http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/, 2017.

[6] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proc. of ACM SOSP*, pp. 85–100, ACM, 2011.

[7] R. Curtmola, J. A. 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.

[8] 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 *Proc. of CRYPTO*, Springer, 2013.

[9] B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin, "Malicious-client security in blind seer: A scalable private dbms," in *Proc. of IEEE S&P*, 2015.

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

[11] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. of IEEE ICDE*, 2012.

[12] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[13] A. Boldyreva and N. Chenette, "Efficient fuzzy search on encrypted data," in *Proc. of FSE*, 2014.

[14] X. Yuan, H. Cui, X. Wang, and C. Wang, "Enabling privacy-assured similarity retrieval over millions of encrypted records," in *Proc. of ESORICS*, 2015.

[15] A. Andoni and P. Indyk, "Efficient algorithms for substring near neighbor problem," in *Proc. of ACM SODA*, 2006.

[16] R. Bost, "Forward secure searchable encryption," in *Proc. of ACM CCS*, 2016.

[17] E.-J. Goh, "Secure indexes," *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[18] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. of ACM CCS*, 2012.

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

[20] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu, "Enabling search over encrypted multimedia databases," in *Proc.of SPIE Media Forensics and Security*, 2009.

[21] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *Proc. of VLDB*, 2007.

[22] X. Yuan, X. Wang, C. Wang, Q. Chen, and J. Lin, "Building an encrypted, distributed, and searchable key-value store," in *Proc. of ACM AsiaCCS*, 2016.

[23] R. Bost, P.-A. Fouque, and D. Pointcheval, "Verifiable Dynamic Symmetric Searchable Encryption: Optimality and Forward Security." Cryptology ePrint Archive, Report 2016/062, 2016.

[24] 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," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205–220, 2007.