Efficient Similarity Search over Encrypted Data

Mehmet Kuzu, Mohammad Saiful Islam, Murat Kantarcioglu

Department of Computer Science, The University of Texas at Dallas
Richardson, TX 75080, USA

$\{mehmet.kuzu, saiful, muratk\}$ @ utdallas.edu

***Abstract*— In recent years, due to the appealing features of cloud computing, large amount of data have been stored in the cloud. Although cloud based services offer many advantages, privacy and security of the sensitive data is a big concern. To mitigate the concerns, it is desirable to outsource sensitive data in encrypted form. Encrypted storage protects the data against illegal access, but it complicates some basic, yet important functionality such as the search on the data. To achieve search over encrypted data without compromising the privacy, considerable amount of searchable encryption schemes have been proposed in the literature. However, almost all of them handle exact query matching but not similarity matching; a crucial requirement for real world applications. Although some sophisticated secure multi-party computation based cryptographic techniques are available for similarity tests, they are computationally intensive and do not scale for large data sources.**

In this paper, we propose an efficient scheme for similarity search over encrypted data. To do so, we utilize a state-of-the-art algorithm for fast near neighbor search in high dimensional spaces called locality sensitive hashing. To ensure the confidentiality of the sensitive data, we provide a rigorous security definition and prove the security of the proposed scheme under the provided definition. In addition, we provide a real world application of the proposed scheme and verify the theoretical results with empirical observations on a real dataset.

I. Introduction

In today's data intensive environment, cloud computing becomes prevalent due to the fact that it removes the burden of large scale data management in a cost effective manner. Hence, huge amount of data, ranging from personal health records to e-mails, are increasingly outsourced into the cloud. At the same time, transfer of sensitive data to untrusted cloud servers leads to concerns about its privacy. To mitigate the concerns, sensitive data is usually outsourced in encrypted form which prevents unauthorized access. Although encryption provides protection, it significantly complicates the computation on the data such as the fundamental search operation. Still, cloud services should enable efficient search on the encrypted data to ensure the benefits of a full-fledged cloud computing environment. In fact, sizable amount of algorithms have been proposed to support the task which are called searchable encryption schemes [1]–[8]. Traditionally, almost all such schemes have been designed for exact query matching. They enable selective retrieval of the data from the cloud according to the existence of a specified feature. In real world, however, it is more natural to perform retrieval according to the similarity with the specified feature instead of the existence of it.

A similarity search problem consists of a collection of data items that are characterized by some features, a query that specifies a value for a particular feature and a similarity metric to measure the relevance between the query and the data items. The goal is to retrieve the items whose similarity against the specified query is greater than a predetermined threshold under the utilized metric. Although exact matching based searchable encryption methods are not suitable to achieve this goal, there are some sophisticated cryptographic techniques that enable similarity search over encrypted data [9], [10]. Unfortunately, such secure multi-party computation based techniques incur substantial computational resources. According to a recent survey [11], secure edit distance computations [10] require over two years to compute similarity between two datasets of 1000 strings each, on a commodity server. It is apparent that we need efficient methods to perform similarity search over large amount of encrypted data. In this paper, we propose a secure index based encryption scheme to meet this requirement.

The basic building block of our secure index is the state-of-the-art approximate near neighbor search algorithm in high dimensional spaces called locality sensitive hashing (LSH) [12]. LSH is extensively used for fast similarity search on plain data in information retrieval community (e.g., [13]). In our scheme, we propose to utilize it in the context of the encrypted data. In such a context, it is critical to provide rigorous security analysis of the scheme to ensure the confidentiality of the sensitive data. In fact, we provide a strong security definition and prove the security of the proposed scheme under the provided definition. In addition, we provide a real world application of our scheme and verify the theoretical results with empirical analysis. In summary, there are several notable contributions of this paper:

Secure LSH Index : To utilize the appealing properties of LSH in the context of the encrypted data, we propose a secure LSH index and a similarity searchable symmetric encryption scheme on top of this index. In addition, we adapt the widely accepted adaptive semantic security definition of Curtmola et. al. [4] for searchable symmetric encryption schemes and prove the security of the proposed scheme under the adapted definition.

Fault Tolerant Keyword Search : We provide an important application of the proposed scheme for fault tolerant keyword search over encrypted data. Typographical errors are common both in the search queries and the data sources, but most of the available searchable encryption schemes do not tolerate such errors. Recently, a fuzzy keyword set based scheme has been proposed to handle the problem [5]. Although the approach of [5] provides a solution to some extent, it is specific to a particular distance measure. On the other hand, our scheme provides more generic solution and it can be utilized for distinct similarity search contexts where LSH is applicable.

Separation of Leaked Information : Almost all the practical searchable encryption schemes leak some information such as the identifiers of encrypted items corresponding to the trapdoor of a search query. Such leakage may lead to statistical attacks if an adversary has some background knowledge. To mitigate the risk, we propose a two server setting to separate the leaked information. Multi-server setting also enables an alternative encryption scheme with lighter clients by transferring more computational burden to the servers.

The remainder of the paper is organized as follows. Section II formulates the problem. Section III provides background information and the security definition. Section IV presents the proposed solution. We present a potential application of the proposed scheme as a case study in Section V and report our experimental analysis in Section VI. We review related work in Section VII and conclude in Section VIII.

II. Problem Formulation

Suppose Alice has a set of sensitive data that she wants to outsource to a cloud server owned by Bob. Due to the confidentiality, data is stored in encrypted form on the remote server in such a way that Bob cannot infer any useful information about the data except for the one Alice allows to leak. In this setting, permitted data users should be able to selectively retrieve items from the remote server. To do so, server should be able to search over encrypted data and return the items that are most similar to the user's request in a reasonable amount of time. Efficient similarity search capability is provided through a secure index.

Definition II.1 Similarity Searchable Symmetric Encryption: Let D be a collection of sensitive data, F_i be a feature set that identifies D_i where $D_i \in D$, F be the set of all possible features, C_i be the encrypted form of D_i, and I be the secure index. Then, similarity searchable symmetric encryption scheme is described as follows:

Keygen(s) : Given a security parameter ψ, outputs a private key K such that $K \in \{0,1\}^{\psi}$.

Enc(K, D_i) : Encrypts D_i with key K and outputs C_i.

Dec(K, C_i) : Decrypts C_i with key K and outputs D_i.

BuildIndex(K, D) : Extracts feature set F_i for each $D_i \in D$ and outputs I which is constructed with the extracted features.

Trapdoor(K, f) : Generates a trapdoor for a specific feature $f \in F$ with key K and outputs trapdoor T.

Search(I, T) : Performs search on I according to the provided trapdoor of the feature f (T) and outputs encrypted data collection C. Suppose that F_j is the feature set associated with D_j. Then, $C_j \in C$ iff the similarity between f and some feature in F_j is greater than a predefined threshold under the utilized similarity metric.

Note that, the definition of Search does not assume any categorization for the features, but consider all the features as a member of the same category. For instance, consider a document as a sensitive data item, words in the document as a feature set and a query keyword as a specific feature. Clearly, similarity searchable symmetric encryption scheme can easily be achieved by utilizing available secure multi-party computation protocols which enable distance calculation between encrypted objects [9], [10]. However, such protocols do not scale for real data sources due to the intensive computational requirements. To enable efficient similarity search, some approximation algorithms (e.g., LSH) are widely used for plain data. We can utilize such efficient methods in the context of encrypted data. To do so, we need to relax the definition of search operation. Let $dist : F \times F \mapsto \mathcal{R}$ be a function that gives the distance between two features, α and β be the thresholds for the utilized similarity metric s.t. $\alpha < \beta$. Then, FuzzySearch is defined as follows:

FuzzySearch(I, T) : Performs search on I according to provided trapdoor of the feature f (T) and outputs encrypted data collection C. Suppose F_j is the feature set associated with D_j. Then, with high probability, $C_j \in C$ if $\exists f_i$ $(dist(f_i, f) \leq \alpha)$ and $C_j \notin C$ if $\forall f_i$ $(dist(f_i, f) \geq \beta)$ where $f_i \in F_j$.

Although *FuzzySearch* is not as precise as *Search* in terms of the characteristics of the retrieved data, it enables very efficient similarity searchable symmetric encryption scheme.

III. Background and Definitions

The basic building block of our solution is locality sensitive hashing (LSH). We present an overview of LSH in Part III-A and we continue with the security definition in Part III-B.

A. Locality Sensitive Hashing

LSH is an approximation algorithm for near neighbor search in high dimensional spaces [12], [14]. The basic idea of LSH is to use a set of hash functions to map objects into several buckets such that similar objects share a bucket with high probability, while dissimilar ones do not. LSH uses locality sensitive function families to achieve this goal.

Definition III-A.1 ($\mathbf{r_1, r_2, p_1, p_2}$)-sensitive Family: Let r_1, r_2 be distances according to the utilized distance metric ($dist : F \times F \mapsto \mathcal{R}$), such that $r_1 < r_2$ and p_1, p_2 be probabilities such that $p_1 > p_2$. Then, family of hash functions H is said to be (r_1, r_2, p_1, p_2)-sensitive if for any $x, y \in F$ and for any $h \in H$

- if $dist(x, y) \leq r_1$, then $Pr[h(x) = h(y)] \geq p_1$.
- if $dist(x, y) \geq r_2$, then $Pr[h(x) = h(y)] \leq p_2$.

LSH is useful for similarity search if p_1 is much higher than p_2 for some application specific r_1 and r_2. This is because, search result should contain almost all the items that are close to the query point and it should not contain more than a reasonable amount of dissimilar items. Fortunately, we can easily construct such locality sensitive functions from a known family via simple constructions [15]. Suppose we are provided with a (r_1, r_2, p_1, p_2)-sensitive family H, we can form a locality sensitive function $g : F \mapsto (g_1(F), ..., g_\lambda(F))$ from

H with an AND-construction followed by an OR-construction. The AND-construction results in a composite hash function $g_i = (h_{i_1} \wedge ... \wedge h_{i_k})$ which is the combination of k random functions from H. In this context, $g_i(x) = g_i(y)$ if and only if $\forall j(h_{i_j}(x) = h_{i_j}(y))$ where $1 \leq j \leq k \ \wedge \ h_{i_j} \in H$. The OR-construction is formed with λ different AND-constructions such that $g(x) = g(y)$ if and only if $\exists i(g_i(x) = g_i(y))$ where $1 \leq i \leq \lambda$. With such a construction, we can turn (r_1, r_2, p_1, p_2)-sensitive family into a (r_1, r_2, p_1', p_2')-sensitive family where $p_1' = 1-(1-{p_1}^k)^\lambda$ and $p_2' = 1-(1-{p_2}^k)^\lambda$ (we refer the reader to [15] for further details). We could then amplify the gap between probabilities by adjusting k and λ to push p_1' closer to 1 and p_2' closer to 0.

B. Security Definition

Over the years, many secure index based protocols have been proposed for searchable symmetric encryption (SSE). SSE schemes are used for finding exact matches corresponding to a query and almost all practical SSE schemes leak some information such as the search and access patterns for efficiency.

Definition III-B.1 Search Pattern (π): Let $\{f_1, ..., f_n\}$ be the feature set for n consecutive queries, then π be a binary matrix s.t. $\pi[i,j] = 1$ if $f_i = f_j$ and $\pi[i,j] = 0$ otherwise.

Definition III-B.2 Access Pattern ($\mathbf{A_p}$): Let $D(f_i)$ be a collection that contains the identifiers of data items with feature f_i and $\{T_1, ..., T_n\}$ be the trapdoors for the query set $\{f_1, ..., f_n\}$. Then, access pattern for the n trapdoors is defined as $\{A_p(T_1) = D(f_1), ..., A_p(T_n) = D(f_n)\}$.

In our scheme, we try to achieve similarity SSE instead of a standard SSE. To achieve this, we extract subfeatures from each feature via LSH. Then, we construct multi-component trapdoors such that a single component is formed for each subfeature. Number of common components between distinct trapdoors may leak relative similarity between them. This kind of leakage for multi-component trapdoors is captured by our new definition called similarity pattern.

Definition III-B.3 Similarity Pattern ($\mathbf{S_p}$): Let $({f_i}^1, ..., {f_i}^y)$ be the subfeatures of f_i, $\{({f_1}^1, ..., {f_1}^y), ..., ({f_n}^1, ..., {f_n}^y)\}$ be the multi-component feature set for n consecutive queries and $i[j]$ represents the j^{th} component of i^{th} feature. Then, $S_p[i[j], p[r]] = 1$ if ${f_i}^j = {f_p}^r$ and $S_p[i[j], p[r]] = 0$ otherwise for $1 \leq i, p \leq n$ and $1 \leq j, r \leq y$.

Security definitions that allow the leakage of only the search and access patterns have been proposed in the context of SSE. Among such definitions, simulation based adaptive security definition of [4] is the widely accepted one in the literature. We adapt this definition for our similarity SSE such that similarity pattern is leaked instead of the search pattern. This is reasonable, because we are trying to achieve similarity matching. For the definition, we need some auxiliary notions that represents the interaction between the client and the server.

Definition III-B.4 History ($\mathbf{H_n}$): Let D be the data collection and $Q = \{f_1, ..., f_n\}$ be the features for n consecutive queries. Then, $H_n = (D, Q)$ is defined as a n-query history.

Definition III-B.5 Trace (γ): Let $C = \{C_1, ..., C_\ell\}$ be the collection of encrypted data items, $id(C_i)$ be the identifier and $|C_i|$ be the size of C_i, $S_p(H_n)$ and $A_p(H_n)$ be the similarity and access patterns for H_n. Then, $\gamma(H_n) = \{(id(C_1), ..., id(C_\ell)), (|C_1|, ..., |C_\ell|), S_p(H_n), A_p(H_n)\}$ is defined as the trace of H_n. Trace is the maximum amount of information that a data owner allows to leak to an adversary.

Definition III-B.6 View (v): Let $C = \{C_1, ..., C_\ell\}$ be the collection of encrypted data items, $id(C_i)$ be the identifier of C_i, I be the secure index and $T = \{T_{f_1}, ..., T_{f_n}\}$ be the trapdoors for H_n. Then, $v(H_n) = \{(id(C_1), ..., id(C_\ell)), C, I, T\}$ is defined as the view of H_n. View is the information that is accessible to an adversary.

Definition III-B.7 Adaptive Semantic Security for Similarity SSE: Similarity SSE scheme is said to be adaptively semantically secure, if there exists a probabilistic polynomial time simulator S for all probabilistic polynomial time adversaries A, such that S can adaptively simulate the adversary's view of the history from the trace with probability negligibly close to 1. Intuitively, this definition implies that all the information that is accessible to the adversary (view) could be constructed from the trace which is essentially the information that data owner allows to leak. Therefore, one can conclude that a similarity SSE scheme does not leak any information beyond the trace if the definition is satisfied by the scheme. More formally, let H_n be a random history from all possible histories, $v(H_n)$ be the view and $\gamma(H_n)$ be the trace of H_n. Then, similarity SSE scheme satisfies the security definition if one can define a simulator S such that for all polynomial size distinguishers $Dist$, for all polynomials $poly$ and a large r:

$$Pr[Dist(v(H_n)) = 1] - Pr[Dist(S(\gamma(H_n))) = 1] < \frac{1}{poly(r)}$$

where probabilities are taken over H_n and the internal coins of key generation and encryption.

IV. Overview of the Solution

In this section, we provide an overview of our solution. To enable efficient similarity search, Alice builds a secure index and outsources it to the cloud server along with the encrypted data items. Server performs search on the index according to the queries of the data users without learning anything about the data other than what Alice allows an adversary to learn. In Part IV-A, we present the index structure. In Part IV-B, we describe the search scheme that is built on top of the index. Then, in Part IV-C, we provide a variation of the basic protocol that involves multiple servers for more secure setting.

A. Secure LSH Index

Our similarity SSE scheme is based on a secure index structure that is built through locality sensitive hashing (LSH). LSH maps objects into several buckets such that similar objects collide in some buckets while dissimilar ones do not with high probability. Index structure is constructed on top of this property. Secure LSH index is constructed with the following 4 steps.

1. Feature Extraction : Let D_i be a sensitive data item and $F_i = \{f_{i_1}, ..., f_{i_z}\}$ be the set of features that characterizes D_i. Then, feature extraction step maps D_i to F_i.

2. Metric Space Translation (ρ)**:** Once the features are extracted, we may need to translate features to vectors in some metric space to apply LSH. This is because, we do not always have locality sensitive function family for the utilized similarity metric, but we can find one after metric space translation. For instance, suppose the features are strings and we want to use the edit distance as a similarity metric. However, there is no known locality sensitive function family for the edit distance. In such a case, we can embed strings into the Euclidean space by approximately preserving the relative edit distance between them [16]. Once strings are translated into the Euclidean space, we can make use of the available families that are defined for the Euclidean distance. In summary, let $F_i = \{f_{i_1}, ..., f_{i_z}\}$ be the feature set of D_i. Then, metric space translation step maps features into a vector set such that $\rho(F_i) = \vec{F_i}$ where $\vec{F_i} = \{\vec{f_{i_1}}, ..., \vec{f_{i_z}}\}$.

3. Bucket Index Construction : In step 3, we apply LSH on the vectors of step 2. Suppose that we have a locality sensitive function family H for the utilized similarity metric. Then, we construct a locality sensitive function $g : (g_1, g_2, ..., g_\lambda)$ from H according to the requirements of our application (see Section III-A). We map each feature vector to λ buckets via composite hash functions g_1 to g_λ. Suppose, $g_i(\vec{f_j})$ be the output of g_i on $\vec{f_j}$. Then, $g_i(\vec{f_j})$ is one bucket identifier of the index and the data items that contain feature f_j are the members of it. Bucket content is simply a bit vector of size ℓ where ℓ is the total number of data items to be stored on the server. Suppose, each data item is assigned an identifier from 1 to ℓ and let $id(D_\eta)$ be the identifier of D_η, B_k be a bucket identifier and V_{B_k} be the bit vector for B_k. Then, $V_{B_k}[id(D_\eta)] = 1$ if and only if $g_i(\vec{f_j}) = B_k$ for some $f_j \in D_\eta$, $g_i \in g$ and $V_{B_k}[id(D_\eta)] = 0$ otherwise.

4. Bucket Index Encryption : In this step, we turn LSH index into a secure index by encrypting bucket identifiers and contents. Bucket identifiers should be encrypted such that only the data users are able to generate them during the query process. Otherwise, an adversary could simply apply the LSH steps to find out the buckets of a specific feature. Similarly, content of the buckets should be encrypted to hide the leakage of important information (e.g., the number of items in a bucket) which may be utilized by statistical attacks. Let K_{id}, $K_{payload} \in \{0,1\}^\psi$ be the secret keys of size ψ, $Enc_{K_{id}}$ be a pseudorandom permutation[1], $Enc_{K_{paylaod}}$ be a PCPA-secure[2] encryption scheme (e.g., AES in CTR mode), B_k be a bucket identifier, V_{B_k} be the bit vector for B_k and I be the secure index. Then, $[Enc_{K_{id}}(B_k), Enc_{K_{payload}}(V_{B_k})] \in I$. Once encryption is done, we need to add some fake records into the index to hide the number of features in the dataset. Let MAX be the maximum number of features that could occur in a dataset, λ be the number of applied composite hash functions, c be the number of records in I, bId_{size} and $bVec_{size}$ be the size of the encrypted bucket identifiers and payloads respectively. Then, addition of $MAX \cdot \lambda - c$ random pairs (R_{i_1}, R_{i_2}) where $|R_{i_1}| = bId_{size}$ and $|R_{i_2}| = bVec_{size}$ prevents the leakage of the number of features[3]. This is because, the number of entries in the index is always equal to $MAX \cdot \lambda$.

Secure LSH index construction is summarized in Alg. 1.

Algorithm 1 Build Index

```
Require: $D$: data item collection, g: $\lambda$ composite hash functions, $\psi$: security
  parameter, $MAX$: maximum possible number of features
  $K_{id} \leftarrow Keygen(\psi)$, $K_{payload} \leftarrow Keygen(\psi)$
  for all $D_i \in D$ do
    $F_i \leftarrow exract\ features\ of\ D_i$
    for all $f_{i_j} \in F_i$ do
      $\vec{f_{i_j}} \leftarrow apply\ metric\ space\ tranlation\ on\ f_{i_j}$
      for all $g_k \in g$ do
        if $g_k(\vec{f_{i_j}}) \notin$ bucket identifier list then
          add $g_k(\vec{f_{i_j}})$ to the bucket identifier list
          initialize $V_{g_k(\vec{f_{i_j}})}$ as a zero vector of size $|D|$
          increment recordCount
        end if
        $V_{g_k(\vec{f_{i_j}})}[id(D_i)] \leftarrow 1$
      end for
    end for
  end for
  for all $B_k \in$ bucket identifier list do
    $V_{B_k} \leftarrow$ retrieve payload of $B_k$
    $\pi_{B_k} \leftarrow Enc_{K_{id}}(B_k)$, $\sigma_{V_{B_k}} \leftarrow Enc_{K_{payload}}(V_{B_k})$
    add $(\pi_{B_k}, \sigma_{V_{B_k}})$ to $I$
  end for
  add $MAX \cdot \lambda - recordCount$ fake records to $I$
  return I
```

B. Basic Secure Search Scheme

In this part, we describe the basic protocol for our similarity SSE scheme, overview of which is presented in Figure 1.

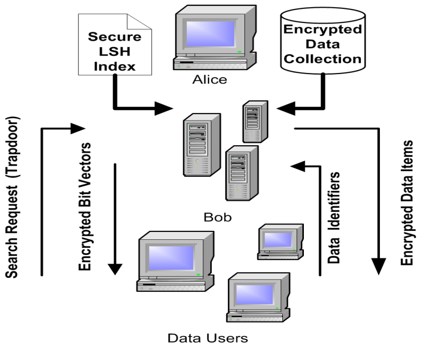

Fig. 1: Basic Secure Search Protocol

1. Key Generation : Initially, Alice generates private keys K_{id}, $K_{payload}$ and K_{coll}.

2. Index Construction : Alice builds an index for the data collection (D) with keys K_{id}, $K_{payload}$ (see Section IV-A).

[1]A random instance of the function family $G : F \mapsto F$ is said to be a pseudorandom permutation if it is computationally indistinguishable from a totally random permutation on F [17].

[2]Pseudorandomness against chosen plaintext attacks (PCPA) guarantees the computational indistinguishability of ciphertexts from random values [4].

[3]Random record addition to keep the index size constant was initially proposed in [4]. We apply the same technique on the LSH buckets.

3. Data Encryption : Alice encrypts the items in D with key K_{coll} to form the encrypted collection E_D. Suppose, $D_i \in D$ and $id(D_i)$ be the identifier of D_i. Then, $(id(D_i), Enc_{K_{coll}}(D_i)) \in E_D$ where $Enc_{K_{coll}}$ is a PCPA-secure[2] encryption.

Alice sends the encrypted collection along with the secure index to the remote server owned by Bob. Once data is outsourced, data users should be able to selectively retrieve data from the remote server. To do so, Alice shares the following information with data users:

- K_{coll} : secret key of data collection encryption
- K_{id}, $K_{payload}$: secret keys of index construction
- ρ : metric space translation function of index construction
- g : locality sensitive hash function of index construction

Shared information enables data users to perform similarity search on the encrypted collection.

4. Trapdoor Construction : Suppose a user is interested in retrieving the items that contain feature f_i. She initially applies metric space translation on f_i ($\vec{f_i} = \rho(f_i)$). Then, she applies LSH to construct the plain query $(g_1(\vec{f_i}), ..., g_\lambda(\vec{f_i}))$ where $g_i \in g$. Finally, she applies pseudorandom permutation $Enc_{K_{id}}$ on each component of the plain query such that $T_{f_i} = (Enc_{K_{id}}(g_1(\vec{f_i})), ..., Enc_{K_{id}}(g_\lambda(\vec{f_i})))$. Once the trapdoor T_{f_i} is constructed, user sends it to the server.

5. Search : Server performs search on the index for each component of T_{f_i} and sends back the corresponding encrypted bit vectors. Suppose, $\{\pi_{B_1}, ..., \pi_{B_\lambda}\}$ is the components of T_{f_i}. Then, the server sends back the set $E_V = \{\sigma_{V_{B_i}} \mid (\pi_{B_i}, \sigma_{V_{B_i}}) \in I \ \wedge \ 1 \leq i \leq \lambda\}$. Once the user receives E_V, she decrypts encrypted bit vectors and ranks the data identifiers.

Item Ranking : Suppose, $S(T_{f_i}) = \{V_{B_k} \mid V_{B_k} = Dec_{K_{payload}}(\sigma_{V_{B_k}}) \ \wedge \ \sigma_{V_{B_k}} \in E_V\}$ is the set of plain bit vectors corresponding to trapdoor T_{f_i}, let $id(D_j)$ be the identifier of D_j and Z_j be a subset of $S(T_{f_i})$. Then, the score of $id(D_j)$ against the trapdoor T_{f_i} is defined as follows:

$$\begin{aligned} score(id(D_j)) \quad &= \quad |Z_j|, \\ Z_j \subset S(T_{f_i}) \quad s.t. \quad & V_{B_k} \in Z_j \quad iff \quad V_{B_k}[id(D_j)] = 1 \end{aligned}$$

Note that, features of D_j are mapped to some buckets during the index construction phase. Similarly, queried feature f_i is mapped to some buckets during the search phase. $score(id(D_j))$ is simply the number of common buckets between the buckets of D_j and f_i. If D_j and f_i shares more buckets, then D_j becomes a stronger candidate for retrieval due to the properties of locality sensitive hashing. Suppose, f_i and f_j are features, metric space vectors of which are mapped to some buckets with the hash functions of g (g_1 to g_λ). Let X be a random variable that represents the number of common buckets (n_b) for f_i and f_j and p be the probability such that $p = Pr[g_k(\vec{f_i}) = g_k(\vec{f_j})]$ where $g_k \in g$. Then,

$$Pr(X = n_b) = \binom{\lambda}{n_b} p^{n_b}(1-p)^{\lambda - n_b}$$

Note that, $Pr(X = n_b)$ grows for larger n_b's with increasing p. Locality sensitive functions are constructed in such a way that p becomes larger with increasing similarity between f_i and f_j. Therefore, large number of common buckets implies high similarity between compared features. Due to this property, if a feature in a data item is highly similar to the queried feature, the score of the corresponding data item is expected to be higher. We can use this expectation to minimize the retrieval of irrelevant items. Only the items with top t highest scores can be requested from the server instead of all which share at least one bucket with the trapdoor.

After ranking, user sends identifiers of the data items with top t highest scores to the server who sends back the encrypted items corresponding to the provided identifiers.

6. Data Decryption : Once the encrypted items corresponding to the search request are retrieved, user decrypts them with the key K_{coll} to obtain their plain versions.

The search procedure is summarized in Algorithms 2 and 3.

Algorithm 2 Search Round 1

```
CLIENT:
Require: $\rho$: metric space translation function, g: $\lambda$ composite hash functions,
  $K_{id}$: secret key, $f$: feature to search
  $\vec{f} \leftarrow \rho(f)$
  for all $g_k \in g$ do
    $\pi_k \leftarrow Enc_{K_{id}}(g_k(\vec{f}))$
  end for
  Trapdoor $T_f \leftarrow (\pi_1, ..., \pi_\lambda)$
  Send $T_f$ to Server

SERVER:
Require: $I : index$, $T_f$: trapdoor
  for all $\pi_k \in T_f$ do
    if $(\pi_k, \sigma_{V_k}) \in I$ then
      Send $\sigma_{V_k}$ to Client
    end if
  end for
```

Theorem 1: Provided similarity SSE scheme is adaptively semantically secure according to Definition III-B.7.

Proof: We will show the existence of polynomial size simulator S s.t. the simulated view $v_S(H_n)$ and the real view $v_R(H_n)$ of a n query history H_n are computationally indistinguishable. Let $v_R(H_n) = \{(id(C_1), ..., id(C_\ell)), (C_1, ..., C_\ell), I, (T_{f_1}, ..., T_{f_n})\}$ be the real view and $\gamma(H_n) = \{(id(C_1), ..., id(C_\ell)), (|C_1|, ..., |C_\ell|), S_p(H_n), A_p(H_n)\}$ be the trace of H_n. Then, S adaptively generates the simulated view $v_S(H_n) = \{(id(C_1)^*, ..., id(C_\ell)^*), (C_1{}^*, ..., C_\ell{}^*), I^*, (T_{f_1}{}^*, ..., T_{f_n}{}^*)\}$ as follows:

- Identifiers of the items are available in the trace. S can simply copy the identifier list, that is, $\{id(C_1)^* = id(C_1), ..., id(C_\ell)^* = id(C_\ell)\}$. Identifier lists of $v_S(H_n)$ and $v_R(H_n)$ are identical, thus they are indistinguishable.
- S chooses n random values $\{C_1{}^*, ..., C_\ell{}^*\}$ such that $|C_1{}^*| = |C_1|, ..., |C_\ell{}^*| = |C_\ell|$. By the definition of PCPA-security [4], output of a PCPA-secure encryption scheme is computationally indistinguishable from a random value. Hence, $C_i{}^*$ and $C_i = Enc_{K_{coll}}(D_i)$ where $Enc_{K_{coll}}$ is a PCPA-secure encryption are computationally indistinguishable.

Algorithm 3 Search Round 2

```
CLIENT:
Require: $E_V$: encrypted bit vectors, $K_{payload}$: secret key, $t$: score limit
  for all $\sigma_{V_{B_k}} \in E_V$ do
    $V_{B_k} \leftarrow Dec_{K_{payload}}(\sigma_{V_{B_k}})$
    for $i \leftarrow 1$ to $|V_{B_k}|$ do
      if $V_{B_k}[i] = 1$ then
        add $i$ to the candidate identifier list
        increment $score(i)$
      end if
    end for
  end for
  idList ← select identifiers from candidates with top t highest scores
  Send idList to Server

SERVER:
Require: $L_{id}$: list of data identifiers, $E_D$: encrypted data collection
  for all $id \in L_{id}$ do
    if $(id, \phi_{id}) \in E_D$ then
      Send $\phi_{id}$ to Client
    end if
  end for

CLIENT:
Require: $C$: encrypted data items, $K_{coll}$: secret key
  for all $\phi_{id} \in C$ do
    $item \leftarrow Dec_{K_{coll}}(\phi_{id})$
  end for
```

- Let bId_{size} and $bVec_{size}$ be the size of an encrypted bucket identifier and an encrypted bit vector respectively, MAX be the maximum number of features that can occur in the user's dataset and λ be the number of components in a trapdoor. Then, S chooses $MAX \cdot \lambda$ random pairs (R_{i_1}, R_{i_2}) s.t. $|R_{i_1}| = bId_{size}$ and $|R_{i_2}| = bVec_{size}$ and inserts them into I^*. Suppose, $(\pi_s, \sigma_{V_s}) \in I$ and $(R_{s_1}, R_{s_2}) \in I^*$, then they are computationally indistinguishable. This is because, $\pi_s = Enc_{K_{id}}(s)$, $\sigma_{V_s} = Enc_{K_{payload}}(V_s)$ and outputs of a pseudorandom permutation and a PCPA-secure encryption are computationally indistinguishable from random values [4], [17]. In addition, both I and I^* contain $MAX \cdot \lambda$ records by the construction. Computational indistinguishability of any single record pair and the equality of the number of indistinguishable records implies computational indistinguishability of I and I^*.
- S simulates trapdoors $\{T_{f_1}, ..., T_{f_n}\}$ in sequence with the help of similarity pattern (S_p). Let $T_{f_i} = \{\pi_{i_1}, ..., \pi_{i_\lambda}\}$ be a multi-component trapdoor with λ components, $T_i[j]$ be the j^{th} component of the i^{th} trapdoor and bId_{size} be the size of an encrypted bucket identifier. If $S_p[i[j], p[r]] = 1$ for any $1 \leq p < i$ and $1 \leq j, r \leq \lambda$, set $T_i[j]^* = T_p[r]^*$. Otherwise, set $T_i[j]^* = R_{i_j}$ where R_{i_j} is a random value such that $|R_{i_j}| = bId_{size}$ and $R_{i_j} \neq T_p[r]^*$ for any $1 \leq p < i$ and $1 \leq r \leq \lambda$. Note that, $T_i[j]$ and $T_i[j]^*$ are computationally indistinguishable since $T_i[j]$ is the output of a pseudorandom permutation and $T_i[j]^*$ is a random value with the same length. In addition, similarity pattern of $\{T_{f_1}, ..., T_{f_n}\}$ is preserved in $\{T_{f_1}{}^*, ..., T_{f_n}{}^*\}$ by the construction. Therefore, simulated and real trapdoor lists are computationally indistinguishable.

Since each component of $v_R(H_n)$ and $v_S(H_n)$ are computationally indistinguishable, we can conclude that the proposed scheme satisfies the security definition presented in III-B.7. ■

We mainly concentrate on the search throughout the paper, but we would like to stress that search and index construction mechanisms could easily be used for the index update. For instance, suppose f and f' be the original and updated versions of a feature in a data item. Alice (data owner) can easily generate the trapdoors for the buckets of f and f' and request the corresponding encrypted bit vectors from the server. Once received bit vectors are updated and encrypted by Alice, they can be sent back to the server. Note that, some buckets of f' may not exist on the server. In such a case, Alice can ask server to replace some fake records with the new real records. To do so, Alice should keep identifiers of the fake records and generate trapdoors with them for replacement request. Similarly, if the content of some buckets for f becomes empty after an update, Alice can ask for replacement of the corresponding records with some fake records via a similar replacement request. Existence of both addition and removal of fake records hides their amount at a particular time.

C. Multi-Server Secure Search Scheme

The basic search scheme reveals the identifiers of the data items corresponding to a particular trapdoor like most of the available practical searchable encryption schemes [4]. However, the security pitfalls of this leakage has not been analyzed in depth in the literature. Hence, hiding association between trapdoors and data identifiers without significant performance degradation is desirable. If we can use two servers instead of a single one, we can easily hide this association. Hence, the protocol becomes resistant to potential statistical attacks that can exploit this leakage.

In the multi-server setting, we assume that both servers are honest but curious and do not collaborate with each other; a common threat model for multi-party protocols. In such a setting, Alice can simply divide the outsourced information between two servers and data users can use the presented basic search scheme with a minor modification as shown in Figure 2-A. To do so, Alice outsources index to Bob and encrypted data collection to Charlie. Then, data users send trapdoors to Bob and receive encrypted bit vectors. Once encrypted bit vectors are decrypted and ranked as described in Part IV-B, identifiers with top t scores are requested from Charlie who sends back the encrypted data items corresponding to the requested identifiers. In summary, round 1 of the basic search is conducted with Bob and round 2 is conducted with Charlie. In this case, Bob observes only the trapdoors and Charlie observes only the data identifiers. Hence, none of them can associate the trapdoors with the identifiers.

Existence of multiple servers enables us to remove some computational burden of the clients by loading more responsibility to the servers. Generally, servers are more powerful

[4]Although oblivous ram model of Goldreich et. al. [18] prevents any information leakage for searchable encryption, such a mechanism is too costly to be practical on real data sources.

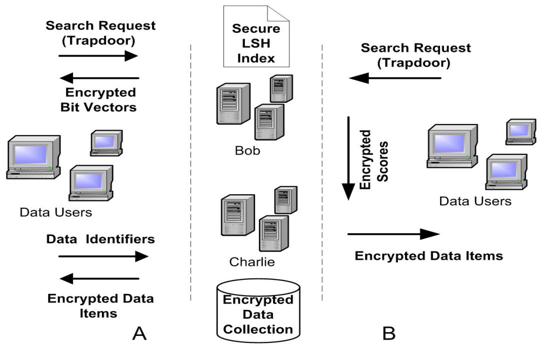

Fig. 2: Multi-Server Secure Search Protocol

than clients in terms of computational resources. Therefore, it is preferable to minimize client computation by transferring the burden to servers as much as possible. To do so, we utilize a semantically secure and additive homomorphic asymmetric encryption scheme called Paillier cryptosystem [19].

If the same message is encrypted multiple times, the ciphers do not equal with very high probability (almost 1). Let, $Enc_{K_{pub}}$ and $Dec_{K_{priv}}$ be the Paillier encryption and decryption functions with keys K_{pub} and K_{priv}, m_1 and m_2 be messages, c_{m_1} and c_{m_2} be ciphers s.t. $c_{m_1} = Enc_{K_{pub}}(m_1)$, $c_{m_2} = Enc_{K_{pub}}(m_2)$. Then, $c_{m_1} \neq c_{m_2}$ even if $m_1 = m_2$ with very high probability. In addition, Paillier cryptosystem has homomorphic additive property. Given the ciphers c_{m_1} and c_{m_2}, there exists an efficient algorithm to compute the encryption of m_1+m_2. $Enc_{K_{pub}}(m_1+m_2) = c_{m_1} \odot_{K_{pub}} c_{m_2}$ where $\odot_{K_{pub}}$ represents the homomorphic addition of two ciphers. Properties of Paillier enables us to construct one round protocol for the client as shown in Figure 2-B. For this protocol, we need a slightly different index structure which we call a Paillier index.

Paillier Index (I') : Let $(\pi_s, \sigma_{V_s}) \in I$ where I is the index of Section IV-A. Then, we could construct I' as follows:

Note that, π_s corresponds to a single bucket and σ_{V_s} is the encrypted bit vector of the bucket. In Paillier index, instead of a single encrypted bit vector, we keep encrypted form of each bit. Suppose, $(\pi_s, \sigma_{V_s}) \in I$. Then, $(\pi_s, [e_{s_1}, ..., e_{s_\ell}]) \in I'$ such that $e_{s_j} = Enc_{K_{pub}}(1)$ if $V_s[id(D_j)] = 1$ and $e_{s_j} = Enc_{K_{pub}}(0)$ otherwise. That is, if D_j contains a feature that is mapped to bucket s by some hash function, then e_{s_j} contains the encrypted bit 1. Otherwise, it contains the encrypted bit 0. Note that, each encryption of zeros and ones are distinct with very high probability (almost 1) due to the fact that Paillier is a semantically secure encryption scheme.

Special structure of the Paillier index enables one round search scheme for the clients. Once index is constructed with Paillier encryption, it is outsourced to Bob along with the Paillier public key. Then, encrypted data collection is outsourced to Charlie along with the Paillier private key. Encryption of data collection is performed as presented in Section IV-B. After Alice outsources the necessary information to the servers, search could be performed as follows:

1. Trapdoor Construction : This step is exactly same as the trapdoor construction step of the basic search scheme. User constructs a multi-component trapdoor $T_{f_i} = \{\pi_1, ..., \pi_\lambda\}$. Then, user sends T_{f_i} to Bob along with a parameter t which will be used for the retrieval of items with top t highest scores.

2. Index Search (Bob) : Once Bob receives T_{f_i}, he extracts the payloads corresponding to all λ components of it and performs homomorphic addition on the encrypted bits. Let $\pi_s \in T_{f_i}$ and $(\pi_s, [e_{s_1}, ..., e_{s_\ell}]) \in I'$. Then, for all data identifiers i, Bob computes :

$$\omega_{score(i)} = e_{1_i} \odot_{K_{pub}} ... \odot_{K_{pub}} e_{\lambda_i}$$

Scores are constructed via homomorphic addition on encrypted bits and represent the number of common buckets between the trapdoor and a particular data item. Once encrypted scores are computed, Bob sends $(i, \omega_{score(i)})$ pairs to Charlie along with the parameter t.

3. Identifier Resolution (Charlie) : Once Charlie receives the necessary information from Bob, he simply decrypts the encrypted scores. Then, he performs a ranking among the data items according to their scores. Ranking is performed exactly the same way as the ranking step of basic search (see Section IV-B). After ranking, Charlie sends back the encrypted data items with top t highest scores among the ones with a score of at least 1.

Algorithm 4 One Round Multi-Server Search

```
CLIENT :
Require: $\rho$: metric space translation function, g: $\lambda$ composite hash functions,
  $K_{id}$: secret key, $f$: feature to search, $t$: score limit
  $\vec{f} \leftarrow \rho(f)$
  for all $g_s \in g$ do
    $\pi_s \leftarrow Enc_{K_{id}}(g_s(\vec{f}))$
  end for
  Trapdoor $T_f \leftarrow (\pi_1, ..., \pi_\lambda)$
  Send $(T_f, t)$ to Bob

SERVER (Bob):
Require: $I'$ : index, $T_f$: trapdoor, $K_{pub}$ : Paillier public key, $t$ : score limit
  for all $i \in$ identifier list of data collection do
    for all $\pi_s \in T_f$ do
      if $(\pi_s, [e_{s_1}, ..., e_{s_i}, ...e_{s_\ell}]) \in I'$ then
        $\omega_{score(i)} \leftarrow \omega_{score(i)} \odot_{K_{pub}} e_{s_i}$
      end if
    end for
  end for
  Send $(i, \omega_{score(i)})$ pairs and parameter $t$ to Charlie

SERVER (Charlie):
Require: $S_L$ : encrypted score list, $K_{priv}$: Paillier private key, $E_C$: en-
  crypted data collection
  for all $(id(D_i), \omega_{score(id(D_i))}) \in S_L$ do
    $score(id(D_i)) \leftarrow Dec_{K_{priv}}(\omega_{score(id(D_i))})$
  end for
  Send encrypted data items corresponding to the identifiers with top $t$ highest
  scores to the client
```

One round scheme which is summarized in Alg. 4 removes some burden from the client by assigning a considerable amount of load on servers in terms of storage, computation and bandwidth. However, such extra load may worth it in some scenarios since cloud servers have significantly more resources compared to the clients. For instance, reduction of a little burden may sometimes be very valuable for some clients such as mobile devices. Note that, the computation and the bandwidth consumption of servers are linear in the number of data items but not in the number of features. Hence, one round scheme will be preferable if the number of data items is relatively small for a specific user but the number of features is large. In summary, we can say that there exists a trade-off between one and two round schemes according to the computational resources of clients and servers and the number of data items. We evaluated the computational burden of both one and two round schemes in Section VI.

V. Case Study: Error Aware Keyword Search over Encrypted Data

In this part, we provide a specific application of the proposed similarity searchable encryption scheme to clarify its mechanism. Most of the available searchable encryption schemes for keyword search is designed for exact matching. Hence, they do not tolerate any typographical errors in the queries or data sources which is common for many real world scenarios. Our similarity SSE scheme offers efficient similarity tests and could easily be utilized for fault tolerant keyword search.

In this context, data items be the documents, features of the documents be the words in them and query feature be a keyword. Initially, we need to find a metric for measuring the similarity between strings. Next, we need a locality sensitive function family that enables us to locate similar strings efficiently. To find a similarity metric along with a known locality sensitive function family, we need to apply some translation on the strings. One of the most effective and simplest translation that we could use is to embed strings into Bloom filters. Translation of strings into the Bloom filter encodings is widely used for similarity measurement in the context of private record linkage [20]. After encoding, we can measure the similarity with a set based similarity measure such as Jaccard distance. It is known that there is an effective locality sensitive hash family that is designed for Jaccard measure called Minhash [21]. We could make use of Minhash as a base to construct the locality sensitive function that we need.

Bloom Filter Encoding : A Bloom filter is a bit array that is affiliated with some hash functions [20]. Each function maps a given element to a bit location with a uniform probability. To embed string S into the filter, it is represented as the set of substrings of length n, or n-grams. Each n-gram is then subject to each hash function and the corresponding bit locations are set to 1. A sample encoding is presented in Figure 3, the 2-grams of strings are encoded with two hash functions.

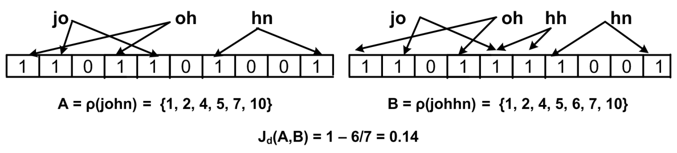

Fig. 3: Sample Bloom Filter Encoding

Once strings are mapped into the Bloom filter, we can make use of Jaccard to measure the distance. Let A and B be two sets, then Jaccard distance between them is defined as follows:

$$J_d(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Bloom filter encodings can easily be represented as sets. If the value of a particular bit location is 1 in the encoding, then that location is the member of the corresponding set. Let $encode$ be the encoding function, L be the list of bit locations in the encoding and ρ be the function that translates strings to sets via Bloom filter encoding. Then ρ is defined as follows:

$$\rho(S) = \{i \mid i \in L \ \wedge \ encode(S)[i] = 1\}$$

We make use of ρ as a metric space translation function in this context. An example translation and Jaccard distance calculation is demonstrated in Figure 3. As mentioned before, there exists an effective locality sensitive family for Jaccard distance called MinHash [21].

MinHash Family : Let Δ be the domain of all possible elements of a set, P be a random permutation on Δ, $P[i]$ be the element in i^{th} position of P and min be a function that returns the minimum of a set of numbers. Then minhash of a set R under P ($h_P(R)$) is defined as :

$$h_P(R) = min(\{i \mid 1 \leq i \leq |\Delta| \ \wedge \ P[i] \in R\})$$

MinHash family (M_F) be then simply the set of minhash functions under distinct random permutations on Δ.

Example V.1 : In Figure 3, metric space translation is applied on 'john' and 'johhn'. In the new space, we can apply minhash functions. Note that, the encodings are 10 bit long. Hence, domain of all possible elements of a set corresponding to an encoding can be specified as $\Delta = \{i \mid 1 \leq i \leq 10\}$. Let $P_1 = \{8,3,6,1,9,5,2,4,7,10\}$ be a permutation on Δ, then $h_{P_1}(\rho(john)) = 4$ and $h_{P_1}(\rho(johhn)) = 3$. This is because, $\{1\}$ is the first element in P_1 which is a member of $\rho(john)$ with location 4 in P_1. Similarly, $\{6\}$ is the first element in P_1 which is a member of $\rho(johhn)$ with location 3 in P_1.

MinHash is shown to be a $(r_1, r_2, 1-r_1, 1-r_2)$-sensitive family for any r_1 and r_2 such that $r_1 < r_2$ (see [15]).

Example V.2 : Let $r_1 = 0.2$ and $r_2 = 0.7$. Then, we have a (0.2, 0.7, 0.8, 0.3)-sensitive family. We can construct a new family from MinHash via AND construction followed by an OR construction as presented in Section III-A. Suppose we need a family $G = \{g : (g_1, ..., g_\lambda)\}$ such that if $J_d(A,B) \leq$ 0.2, then $Pr[\exists i \ (g_i(A) = g_i(B))] \geq 0.99$ and if $J_d(A,B) \geq$ 0.7, then $Pr[\exists i \ (g_i(A) = g_i(B))] \leq 0.01$ where $1 \leq i \leq \lambda$. We can easily obtain such a family from M_F through AND-OR construction with parameters $\lambda = 20$ and $k = 7$. With this setting, we have a (0.2, 0.7, 0.99, 0.004)-sensitive family.

Once similarity metric, metric space translation function and locality sensitive function family are determined, remaining steps are standard for any similarity search. Index construction and search schemes of Sections IV-A and IV-B can easily be applied afterward.

VI. Experimental Evaluation

In this section, we present the experimental evaluation of the proposed scheme. We investigated the success of our scheme in the context of error aware keyword search, but we would like to stress that the scheme can be applied to distinct similarity search contexts over encrypted data. To perform our evaluation, we used a publicly available Email dataset, namely the Enron dataset [22]. We constructed a sample corpus of 5000 e-mails via random selection from Enron. Then, we built a secure LSH index on the sample corpus. For index construction, we embedded 2-grams of words into 500 bit Bloom filter with 15 hash functions as described in Section V. This setting is recommended in [20] for the encoding of the strings.

Note that, we need to determine distance thresholds for our FuzzySearch scheme according to requirements of our application (see Section II). To do so, we perturbed the keywords by introducing typographical errors and measured the Jaccard distance between the encodings of the original and perturbed versions. For perturbation, we used a publicly available typo generator [23] which produces a variety of spelling errors (e.g., skipped, reversed, inserted letters). According to our empirical observations, only a small percentage of the measurements had a distance that is more than 0.8 and most of them had a distance that is less than 0.45. Therefore, we wanted to retrieve a mail M upon query q if $\exists w_i (w_i \in M \ \wedge \ J_d(\rho(w_i), \rho(q)) \leq 0.45)$ and not to retrieve it if $\forall w_i (w_i \in M \ \wedge \ J_d(\rho(w_i), \rho(q)) \geq 0.8)$, where w_i is a word in M and ρ is the encoding function. In summary, we determined FuzzySearch thresholds ($\alpha = 0.45, \ \beta = 0.8$) via some empirical analysis. One can use other methods to determine these thresholds depending on the application context (e.g., domain experts, historical data analysis). We built a locality sensitive family from MinHash according to the thresholds. To do so, we set the parameters of AND-OR construction ($k = 5, \ \ \lambda = 37$) such that we have a (0.45, 0.8, 0.85, 0.01)-sensitive family.

For encryption of both e-mails and secure index constructs, we used AES block cipher in CTR mode [17] with 128 bit secret key. For one round search scheme of multi-server setting, we make use of Paillier index (see Section IV-C). In such setting, Paillier encryption was performed with 512 bit public key. To do so, we used our publicly available Paillier Threshold Encryption toolbox [24]. The retrieval and performance evaluations of our scheme are presented in parts VI-A and VI-B respectively.

A. Retrieval Evaluation :

The experiments of this part have been conducted in a single server setting, but note that the results are completely valid for multi-server setting. To evaluate the retrieval success, we initially selected 1000 random keywords from the corpus. Then, we generated queries by perturbing the selected keywords 25% of the time via the typo generator. In other words, 250 of the queries had some spelling error[5]. We formed trapdoors for the queries by following the trapdoor construction method of Section IV-B with the settings of the index setup (e.g., LSH function). To verify the LSH based theoretical results of the scheme with empirical observation, we calculated average retrieval ratio corresponding to the issued queries.

Average Retrieval Ratio : Let ρ be the encoding function, q be the query, w_q be a word in document D_i s.t. $\rho(w_q)$ is the most similar encoding to $\rho(q)$ among all encodings $\rho(w_j)$ for $w_j \in D_i$. Then, the distance between D_i and q $(dist(q, D_i))$ is defined to be $J_d(\rho(w_q), \rho(q))$ where J_d is the Jaccard distance. Suppose D is the collection of all documents, $R^{D(q)}$ is the set of retrieved documents when query q is issued, $D_{d_k}(q)$ is the set of documents within d_k distance from q s.t. $D_j \in D_{d_k}(q)$ iff $(D_j \in D \ \wedge \ d_k \leq dist(q, D_j) < d_k + \epsilon)$ where ϵ is a small constant and $R^{D_{d_k}(q)} = R^{D(q)} \cap D_{d_k}(q)$. Then, retrieval ratio of query q and average retrieval ratio of the query set $Q = \{q_1, ..., q_n\}$ for distance d_k are defined as:

$$rr_{d_k}(q) = \frac{|R^{D_{d_k}(q)}|}{|D_{d_k}(q)|}, \quad arr_{d_k}(Q) = \frac{\sum_{j=1}^{n} rr_{d_k}(q_j)}{n}$$

Figure 4 shows the average retrieval ratio for the issued queries. As mentioned in the index setup, we formed LSH function to retrieve D_i if $dist(D_i, q) \leq 0.45$ and not to retrieve it if $dist(D_i, q) \geq 0.8$ with high probability. The range [0.45, 0.8] is the fuzzy range such that the probability of retrieval decreases with increasing distance. Note that, probabilities for the constructed locality sensitive function is valid for single word pair. If the similarity of the encoded versions of a query and a single word in a document is less than a threshold, they don't share a bucket with high probability. On the other hand, a document may contain many words and we take into account all of them for the retrieval decision. One can argue that, this may significantly increase the retrieval probability of irrelevant documents. Fortunately, the probabilities of bucket sharing for distinct query, word pairs are not independent but depend on their location in the metric space. If all the words of a document are far from the query, they do not share any bucket with the query most of the time as verified by our experiments.

Another result that is due to the properties of LSH is the correlation of the number of common buckets and the distance between a query and a document (see Section IV-B). The probability of sharing more buckets increases with decreasing distance which enables item ranking for the retrieval. To verify this, we measured the variation of the distance between a document and a query with changing number of common buckets. Let $R_m^{D(q)}$ be the set of document identifiers that are retrieved when query q is issued such that members of $R_m^{D(q)}$ shares m common buckets with q. Then, the distance of the query q and average distance of the query set $Q = \{q_1, ..., q_n\}$ for m common buckets are defined as:

[5]As pointed out in [5], 23% of the time search input "Britney" is misspelled according to Google statistics. Hence, we chose %25 as a reasonable error rate for our experiments.

$$dst_m(q) = \frac{\sum\limits_{D_i \in R_m^{D(q)}} dist(D_i, q)}{|R_m^{D(q)}|}, \; adst_m(Q) = \frac{\sum_{j=1}^{n} dst_m(q_j)}{n}$$

Figure 5 depicts the $adst_m(Q)$ for issued 1000 queries. This result verifies the applied ranking mechanism.

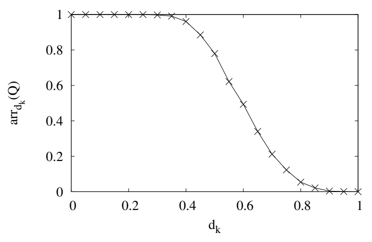

Fig. 4: Distance vs. Avg. Retrieval Ratio

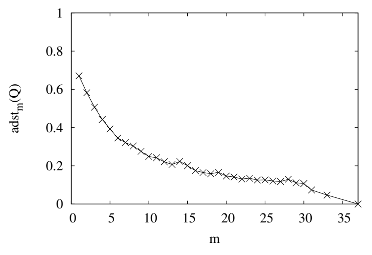

Fig. 5: Number of Common Buckets vs. Avg. Distance

To evaluate the success of the error aware keyword search, we used precision and recall as an evaluation metric. Let $D(w)$ be the set of documents that contain word w and w' be the query issued for retrieving $D(w)$. Note that, 25% of the time $w \neq w'$ due to the perturbation. In this context, we want to retrieve document D_i upon issuing the query w' if and only if $w \in D_i$. Suppose $R^{D(w')}$ be the set of documents that are retrieved when w' is issued and $R^{D(w)}$ be the subset of $R^{D(w')}$ such that members of it contain w. Then, the precision and recall for w' and the average precision and recall for the set $W' = \{w'_1, ..., w'_n\}$ are defined as follows:

$$prec(w') = \frac{|R^{D(w)}|}{|R^{D(w')}|}, \; aprec(W') = \sum_{j=1}^{n} \frac{prec(w'_i)}{n}$$

$$rec(w') = \frac{|R^{D(w)}|}{|D(w)|}, \; arec(W') = \sum_{j=1}^{n} \frac{rec(w'_i)}{n}$$

Once a query is issued, retrieved document identifiers are ranked according to their scores. Then, items with top t highest scores are requested from the server. Average precision and recall for issued 1000 queries with changing t is demonstrated in Figure 6. As expected, less similar items are retrieved with increasing t.

In addition to similar item retrieval, one may want to retrieve only the exact matches for a particular query. In such a case, she can only request the documents which shares maximum

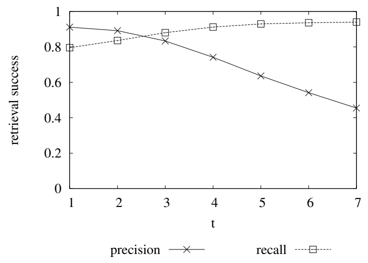

Fig. 6: Error Aware Keyword Search Evaluation

possible number of buckets with the query. In our experimental setting, each query is mapped to 37 buckets, so the query and a document should share 37 buckets for the exact match scenario. We issued 1000 unperturbed queries to evaluate the success of exact match scenario. Average precision and recall are 0.99 and 1 respectively. Hence, one can easily switch to exact match without almost any degradation in the retrieval quality.

Note that, the retrieval results are due to the properties of LSH. Theory behind LSH offers successful retrieval results and one can arrange its parameters (k, λ) according to the requirements of a specific application (see Section III-A). Our similarity searchable symmetric encryption scheme enables secure utilization of LSH.

B. Performance Evaluation :

In this part, we evaluated the performance of both basic and one round multi-server search schemes. Note that, the computational requirement of two round multi-server scheme is almost same as the basic scheme from the user's perspective. For the basic scheme, we evaluated the effect of LSH parameters (k and λ), number of data items (n_d) and number of features (n_f) on the search performance. To do so, we measured the average search time and transferred protocol data for 1000 queries with distinct settings in our local network. Search time is simply the time between the search request and the identification of the data identifiers. Protocol data is the amount of the data transferred between the client and server for the protocol (e.g., trapdoors + encrypted bit vectors + document identifiers[6]). It does not include the final transfer of the requested items which highly depends on the size of the data items but not the protocol. To observe the effect of distinct settings, we modified a single parameter at a time and used the default values for the others. We used $k = 5$, $\lambda = 37$, $n_d = 3000$ and $n_f = 3000$ as default values[7]. In error aware keyword search context, n_d is the number of documents and n_f is the number of distinct words that are indexed. That is, we selected n_f random words and n_d random documents from our sample corpus. Then, we indexed the selected documents using only the selected words.

[6]In worst case, one can request all the documents that shares at least 1 bucket with the query. Hence, identifiers in the protocol data include all such identifiers instead of the top ranked ones.

[7]Index construction with default settings could be done in a few minutes on a commodity server and it can easily be parallelizable if necessary.

Both protocol data and search time decreases with increasing k, as shown Figure 7-a. This is because, less number of data items are identified by the search due to the more restricted distance range of larger k. Large k offers retrieval of only very similar items. In addition to less transmission time, ranking of less items on client side decrements the search time. Decrease in k and increase in λ have similar effects in terms of final search results (see Section III-A). Hence, with increasing λ, protocol data and search time increases as shown in Figure 7-b. But note that, effects of k and λ are not parallel. This is because, increase in λ has additional cost of larger trapdoors. Although increase in k does not improve the performance that much after a point (e.g., only exact matches are retrieved), decrease in λ does due to the smaller trapdoors. The effect of changing n_d and n_f is demonstrated in Figures 7-c and 7-d respectively. Increase in the number of documents results in more items that satisfies the search request. In addition, encrypted bit vectors that are transferred to the client becomes larger due to the fact that vector contains n_d bits (see Section IV-A). On the other hand, increment in n_f does not change the performance that much as expected. Addition of more terms into the index slightly augments matching items against a query which in turn causes a minor decrease in performance.

In addition to basic and two round multi-server search schemes, we proposed Paillier index based one round multi-server search scheme. Although one round scheme mitigates the computational burden of the clients, it brings extra burden to the servers which is linear in the number of data items (n_d). Hence, such scheme is useful if n_d is limited but the number of features (n_f) is high (e.g., large documents, videos). The main bottleneck of one round scheme is the transfer of homomorphic addition results between two servers which depends on n_d (see Section IV-C). In Figure 8-a, we presented the change in search time and transferred protocol data between two servers with changing n_d. As expected, the protocol data and the search time increases linearly with increasing n_d. On the other hand, search performance does not depend on n_f as demonstrated in Figure 8-b. In summary, one round scheme is a practical setting for a limited number of data items with large set of features. If the number of data items is huge, then it is more practical to go with two round schemes.

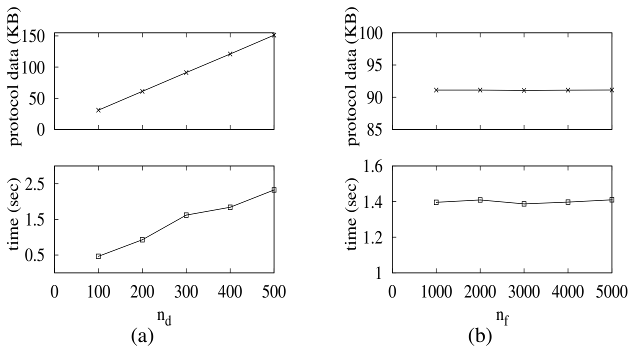

Fig. 8: One Round Scheme Search Performance

VII. Related Work

Over the years, various protocols and security definitions have been proposed for searchable encryption. Optimal security is achieved by the oblivous RAM model [18] of Goldreich et. al. which does not leak any information to the server. However, this model is impractical for real world scenarios due to the excessive computational costs (e.g., polylogarithmic rounds of interaction for each search). As an alternative to heavyweight oblivious RAM, there are approaches ([1]–[8]) to selectively leak information (e.g., search pattern and access pattern) to provide practical searchable encryption schemes.

The first of such practical approaches was provided in [7]. Song et. al. proposed to encrypt each word of a document with a special encryption construct. Later on, Goh et. al. proposed a security definition to formalize the security requirements of searchable symmetric encryption in [6]. Similarly, Chang et. al. introduced a simulation based security definition in [2] which is slightly stronger than the definition of [6]. However, both definitions do not consider adaptive adversaries which could generate the queries according to the outcomes of previous queries. The shortcomings of [2] and [6] have been addressed in [4], where Curtmola et. al. presented adaptive security definition for searchable encryption schemes. In [4], they also provided a protocol that is compatible with their definition.

All the secure index based schemes presented so far, do not enable similarity matching but only exact matching in the context of keyword search. Hence, such schemes are not resistant to the typographical errors which is common for real world search scenarios. Li et. al. proposed wildcard based fuzzy keyword search scheme over encrypted data in [5]. Although fuzzy scheme tolerates errors to some extent, it is only applicable to strings under edit distance. Also, fuzzy sets may become too big if we have long words which necessitates to issue large trapdoors. Another similarity matching technique for encrypted data was proposed in [25]. Their approach is applicable to approximate string search under Hamming distance. In contrast to this study, their approach assumes categorical division in the data sources. For instance, documents are divided into fields (e.g., email to, from fields) in such a way that each field holds a single value.

Secure index based search was also applied in the context of encrypted multimedia databases [26]. Lu et. al. proposed to extract visual words from images and construct indexes according to them. Their approach is slightly different than traditional search such that query is not a specific feature for an image but all features of an image. The goal is to retrieve images that are similar to the query image.

In addition to index based schemes, there are some sophisticated cryptographic techniques that enable similarity tests. Various distance measures such as edit distance [10] and approximation of Hamming distance [9] can be computed securely. Also, a recent fully homomorphic encryption scheme [27] enables secure computation of various functions on encrypted data. Such techniques can be utilized for similarity

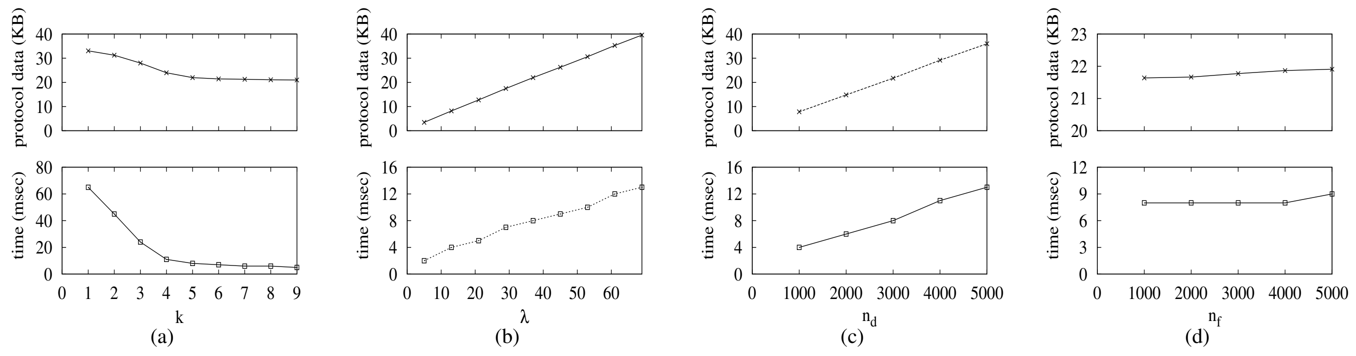

Fig. 7: Basic Scheme Search Performance

search over encrypted data. However, they are inefficient due to their reliance on costly cryptographic operations and they do not scale well for real world data sources.

VIII. Conclusion

In this paper, we proposed an efficient similarity searchable symmetric encryption scheme. To do so, we utilized locality sensitive hashing which is widely used for fast similarity search in high dimensional spaces for plain data. We proposed LSH based secure index and a search scheme to enable fast similarity search in the context of encrypted data. In such a context, it is very critical not to sacrifice the confidentiality of the sensitive data while providing functionality. We provided a rigorous security definition and proved the security of the proposed scheme under the provided definition to ensure the confidentiality. To clarify the properties of the proposed scheme, we presented a real world application of it, namely the error aware keyword search. This application enables keyword search which is tolerant to the typographical errors both in the queries and the data sources. Finally, we illustrated the performance of the proposed scheme with empirical analysis on a real data.

IX. Acknowledgments

This work was partially supported by Air Force Office of Scientific Research MURI Grant FA9550-08-1-0265, National Institutes of Health Grant 1R01LM009989, National Science Foundation (NSF) Grant Career-0845803, and NSF Grants CNS-0964350, CNS-1016343.

References

[1] M. Bellare, A. Boldyreva, and A. ONeill, “Deterministic and efficiently searchable encryption,” in *Proc. of Crypto’07*, 2007, pp. 535–552.
[2] Y. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data,” in *Proc. of ACNS’05*, 2005, pp. 442–455.
[3] D. Boneh and B. Waters, “Conjunctive, subset, and range queries on encrypted data,” in *Theory of Cryptography*, ser. LNCS, 2007, vol. 4392, pp. 535–554.
[4] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: Improved definitions and efficient constructions,” in *Cryptology ePrint Archive, Report 2006/210*, 2006.
[5] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, “Enabling efficient fuzzy keyword search over encrypted data in cloud computing,” in *Cryptology ePrint Archive, Report 2009/593*, 2009.
[6] E. Goh, “Secure indexes,” in *Cryptology ePrint Archive, Report 2003/216*, 2003.
[7] D. Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in *Proc. of the IEEE Symposium on Security and Privacy’00*, 2000, pp. 44–55.
[8] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, “Secure ranked keyword search over encrypted cloud data,” in *Proc. of ICDCS’10*, 2010, pp. 253–262.
[9] J. Feigenbaum, Y. Ishai, K. Nissim, M. Strauss, and R. Wright, “Secure multiparty computation of approximations,” *ACM Transactions on Algorithms*, vol. 2, pp. 435–472, 2006.
[10] M. Atallah, F. Kerschbaum, and W. Du, “Secure and private sequence comparisons,” in *Proc. of the WPES’03*, 2003, pp. 39–44.
[11] E. Durham, Y. Xue, M. Kantarcioglu, and B. Malin, “Quantifying the correctness, computational complexity, and security of privacy-preserving string comparators for record linkage,” *Information Fusion, doi:10.1016/j.inffus.2011.04.004*, 2011.
[12] P. Indyk and R. Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” in 30^{th} *STOC*, 1998, pp. 604–613.
[13] 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’07*, 2007, pp. 253–262.
[14] A. Gionis, P. Indyk, and R. Motwani, “Similarity search in high dimensions via hashing,” in *Proc. of VLDB’99*, 1999, pp. 518–529.
[15] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. http://infolab.stanford.edu/ ullman/mmds/book.pdf, 2010.
[16] C. Faloutsos and K. Lin, “Fastmap: a fast algorithm for indexing, data-mining, and visualization,” in *Proc. of the SIGMOD’95*, 1995, pp. 163–174.
[17] S. Goldwasser and M. Bellare, *Lecture Notes on Cryptography*. http://cseweb.ucsd.edu/ mihir/papers/gb.html, 2008.
[18] O. Goldreich and R. Ostrovsky, “Software protection and simulation on oblivious rams,” *Journal of the ACM*, vol. 43, pp. 431–473, 1996.
[19] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *Proc. of EUROCRYPT’99*, 1999, pp. 223–238.
[20] R. Schnell, T. Bachteler, and J. Reiher, “Privacy-preserving record linkage using Bloom filters,” *BMC Medical Informatics and Decision Making*, vol. 9, no. 1, p. 41, 2009.
[21] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher, “Min-wise independent permutations,” in *Proc. of the* 30^{th} *STOC*, 1998, pp. 327–336.
[22] “Enron email dataset,” http://www.cs.cmu.edu/enron, 2011.
[23] “Typo generator,” https://dbappserv.cis.upenn.edu/spell/, 2011.
[24] “Paillier encryption toolbox,” http://www.utdallas.edu/ mxk093120/cgi-bin/paillier/, 2011.
[25] H. Park, B. Kim, D. H. Lee, Y. Chung, and J. Zhan, “Secure similarity search,” in *Cryptology ePrint Archive, Report 2007/312*, 2007.
[26] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu, “Enabling search over encrypted multimedia databases,” in *Proc.of SPIE Media Forensics and Security’09*, 2009.
[27] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proc. of the* 41^{st} *STOC*, 2009, pp. 169–178.