Privacy-Preserving Multi-Keyword Fuzzy Search over Encrypted Data in the Cloud

Bing Wang* Shucheng Yu† Wenjing Lou* Y. Thomas Hou*
*Virginia Polytechnic Institute and State University, Blacksburg, VA, USA
†University of Arkansas, Little Rock, AR, USA

***Abstract*—Enabling keyword search directly over encrypted data is a desirable technique for effective utilization of encrypted data outsourced to the cloud. Existing solutions provide *multi-keyword exact* search that does not tolerate keyword spelling error, or *single keyword fuzzy* search that tolerates typos to certain extent. The current fuzzy search schemes rely on building an expanded index that covers possible keyword misspelling, which lead to significantly larger index file size and higher search complexity. In this paper, we propose a novel *multi-keyword fuzzy* search scheme by exploiting the locality-sensitive hashing technique. Our proposed scheme achieves fuzzy matching through algorithmic design rather than expanding the index file. It also eliminates the need of a predefined dictionary and effectively supports multiple keyword fuzzy search without increasing the index or search complexity. Extensive analysis and experiments on real-world data show that our proposed scheme is secure, efficient and accurate. To the best of our knowledge, this is the first work that achieves multi-keyword fuzzy search over encrypted cloud data.**

I. Introduction

In cloud computing, scalable and elastic storage and computation resources are provisioned as measured services through the Internet. Outsourcing data services to the cloud allows organizations to enjoy not only monetary savings, but also simplified local IT management since cloud infrastructures are physically hosted and maintained by the cloud providers. To minimize the risk of data leakage to the cloud service providers, data owners opt to encrypt their sensitive data, e.g., health records, financial transactions, before outsourcing to the cloud, while retaining the decryption keys to themselves and other authorized users. This in turn renders data utilization a challenging problem. For example, in order to search some relevant documents amongst an encrypted data set stored in the cloud, one may have to download and decrypt the entire data set. This is apparently impractical when the data volume is large. Thus, mechanisms that allow users to search directly on the encrypted data are of great interest in the cloud computing era.

Since Song et al's seminal work on searchable encryption [1], much effort has been made to design effective and efficient mechanisms to enable search over encrypted data [1]–[14]. Instead of a word-by-word linear scan in the full text search [1], early works [3], [10], [11] built various types of secure index and corresponding index-based keyword matching algorithms to improve search efficiency. All these works only support the search of single keyword. Subsequent works [2], [4]–[7] extended the search capability to multiple, conjunctive or disjunctive, keywords search. However, they support only exact keyword matching. Misspelled keywords in the query will result in wrong or no matching. Very recently, a few works [12]–[14] extended the search capability to approximate keyword matching (also known as *fuzzy search*). These are all for single keyword search, with a common approach involving expanding the index file by covering possible combinations of keyword misspelling so that a certain degree of spelling error, measured by edit distance, can be tolerated. Although a wild-card approach is adopted to minimize the expansion of the resulting index file, for a l-letter long keyword to tolerate an error up to an edit distance of d, the index has to be expanded by $O(l^d)$ times. Thus, it is not scalable as the storage complexity increases exponentially with the increase of the error tolerance. To support multi-keyword search, the search algorithm will have to run multiple rounds.

To date, efficient multi-keyword fuzzy search over encrypted data remains a challenging problem. We want to point out that the efforts on search over encrypted data involve not only information retrieval techniques such as advanced data structures used to represent the searchable index, and efficient search algorithms that run over the corresponding data structure, but also the proper design of cryptographic protocols to ensure the security and privacy of the overall system. Although multi-keyword search and fuzzy search have been implemented separately, a combination of the two does not lead to a secure and efficient multi-keyword fuzzy search scheme. In this paper, we propose a brand new idea for achieving multi-keyword (conjunctive keywords) fuzzy search. Different from existing multi-keyword search schemes, our scheme eliminates the requirement of a predefined keyword dictionary. The fuzziness of the keyword is captured by an innovative data structure and algorithmic design without expanding the keyword index, and hence exhibits a high efficiency in terms of computation and storage. We achieve this by several novel designs based on locality-sensitive hashing (LSH) [15] and Bloom filters [16]. We convert each keyword to its bigram vector representation and utilize Euclidean distance to capture keywords similarity. By constructing file indexes using LSH in Bloom filter, our scheme finds documents with matching keywords efficiently. For ease of presentation, we first present a basic scheme that fulfills the functionality of multi-keyword fuzzy search but has some security vulnerabilities. Based on our basic scheme, we design our enhanced solution by incorporating another

978-1-4799-3360-0/14/$31.00

layer of security protection. Extensive analysis shows that our scheme is secure, efficient and accurate. Experimental results on real-world data validate our claim. Our contributions can be summarized as follows:

1) To the best of our knowledge, this is the first work that addresses the multi-keyword fuzzy search over encrypted data problem with user data privacy protection.
2) In contrast to previous fuzzy keyword search solutions [12]–[14], which require expanded storage for wild-card based fuzzy keyword set, our scheme exploits locality-sensitive hashing to provide efficient fuzzy search with constant size index regardless the number of keywords associated with the file.
3) In contrast to previous solutions on multiple keywords search [2], [4], our scheme eliminates the need of a predefined dictionary, and hence enables efficient file update. Our search process is very efficient - it performs multiple keyword matching in one round by calculating the inner product of two vectors.
4) We implemented our scheme and performed an evaluation using a real-world data set. The results demonstrate that our scheme is accurate and efficient.

The rest of the paper is organized as follows. Section II presents the formulation of our problem as well as the preliminaries. Section III describes our basic privacy-preserving multi-keyword fuzzy search scheme in detail. Based on our basic scheme, we present our enhanced solution in section IV. Section V evaluates our schemes through experimental studies. Section VI discusses the related works. We conclude our paper in section VII.

II. Problem Formulation

We formulate the privacy preserving problem of multiple keyword fuzzy search over encrypted data in this section. We denote a keyword collection of a document as an *index* and an encrypted index as *secure index*. Similarly, a *query* is a keyword collection of a search and a *trapdoor* is an encrypted version of a query.

A. System Model

Fig. 1 shows the overall system architecture. To outsource a set of files to the cloud, the *data owner* builds a secure searchable index for the file set and then uploads the encrypted files, together with the secure index, to the *cloud server*. To search over the encrypted files, an authorized *user* first obtains the trapdoor, i.e, the "encrypted" version of search keyword(s), from the data owner, then submits the trapdoor to the cloud server. Upon receiving the trapdoor, the cloud server executes the search algorithm over the secure indexes and returns the matched files to the user as the search result.

B. Security Model

We adopt the "honest-but-curious" model for the cloud server as in [12]–[14]. It assumes that the cloud server would honestly follow the designated protocols and procedures to fulfill its service provider's role, while it may analyze the

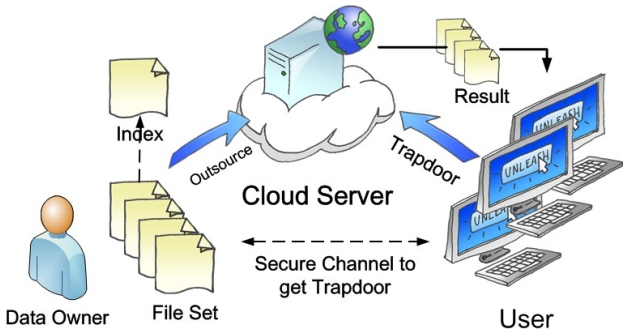

Fig. 1. System architecture of search over encrypted data in cloud computing

information stored and processed on the server in order to learn additional information about its customers. In terms of level of privacy protection, we consider two threat models depending on the information available to the cloud server, which are also used in other related works [8], [17].

- **Known Ciphertext Model:** The cloud server can only access the encrypted files, the secure indexes and the submitted trapdoors. The cloud server can also know and record the search results. The semantic meaning of this threat scenario is captured by the non-adaptive attack model [10].
- **Known Background Model:** The cloud server knows additional background information in this model. The background refers to the information which can be learned from a comparable dataset. For example, the keywords and their statistical information, such as the frequency, obtained from previous years' proceedings of a computer security conference can be very similar with this year's.

We assume users are trusted entities. They have pre-existing mutual trust with the data owner. The obtaining of the trapdoors can be done through regular authentication and secure channel establishment protocols based on the prior security context shared between each user and the data owner. The encryption of the individual files and the distribution of the decryption keys to authorized users are separated issues which have been discussed in other publications [18], [19].

The objective of this scheme is to preserve user data privacy, which includes: 1). file content privacy; 2). index privacy; and 3). user query privacy. While file content privacy can be achieved by encryption-before-outsourcing schemes [18], [19], this paper focuses on preserving the data privacy due to the search functionality and possible information leakage associated with it, as follows,

- **Keyword privacy:** Besides the search result, the cloud server should not deduce any keyword information of the file set from secure indexes and trapdoors. Keyword privacy requires indexes and queries be properly represented and securely encrypted.
- **Trapdoor unlinkability:** The cloud server should not be able to link one trapdoor to another even if they are for the same query. Trapdoor unlinkability requires an non-

deterministic trapdoor generation function.

C. Design Goal

Our design bears the following security and performance goals.

- **Multi-Keyword Fuzzy Search:** Our primary goal is to support multi-keyword fuzzy search. For example, "network security" related files should be found for a mis-spelled query "netword security".
- **Privacy Guarantee:** Our scheme should provide privacy guarantees by not leaking the information about the data files or the query keywords beyond the search results to the cloud server.
- **Result Accuracy:** Since this is about fuzzy search, result accuracy is an important performance metric. Our scheme should find the results as accurate as possible and keep the accuracy within an acceptable range.
- **No Predefined Dictionary:** The need of a pre-defined dictionary is a limiting factor that makes dynamic data operations, such as dataset/index update, very difficult. In our design, we would like to eliminate this requirement in contrast to many previous solutions [2], [4], [8]–[10], [12].

D. Preliminaries

Two important techniques are used in our design, i.e. Bloom filter and locality-sensitive hashing (LSH). Brief introductions are given below.

1) Bloom filter: A Bloom filter is a bit array of m bits, all of which are set to 0 initially. Given a set $\mathcal{S} = \{a_1, a_2, \cdots, a_n\}$, a Bloom filter uses l independent hash functions from $\mathcal{H} = \{h_i | h_i : S \to [1, m], 1 \leq i \leq l\}$ to insert an element $a \in \mathcal{S}$ into the Bloom filter by setting the bits at all the $h_i(a)$-th positions in the array to 1. To test whether an element q is in $\mathcal{S}$, feed it to each of the l hash functions to get l array positions. If the bit at any position is 0, then $q \notin \mathcal{S}$; otherwise, either q belongs to $\mathcal{S}$ or q yields a false positive. The false positive rate of a m-bit Bloom filter is approximately $(1 - e^{-\frac{ln}{m}})^l$. The optimal false positive rate is $(1/2)^l$ when $l = \frac{m}{n} \cdot \ln 2$ [16].

2) Locality-Sensitive Hashing: Given a distance metric d, e.g. Euclidean distance, a LSH function hashes close items to the same hash value with higher probability than the items that are far apart. A hash function family $\mathcal{H}$ is (r_1, r_2, p_1, p_2)-sensitive if any two points s, t and $h \in \mathcal{H}$ satisfy:

$$\text{if } d(s,t) \leq r_1 : Pr[h(s) = h(t)] \geq p_1 \tag{2.1}$$

$$\text{if } d(s,t) \geq r_2 : Pr[h(s) = h(t)] \leq p_2 \tag{2.2}$$

where $d(s,t)$ is the distance between the point s and the point t. We use the p-stable LSH family [20] in our scheme. A p-stable LSH function has the form $h_{a,b}(\mathbf{v}) = \left\lfloor \frac{\mathbf{a} \cdot \mathbf{v} + b}{w} \right\rfloor$ where $\mathbf{a}, \mathbf{v}$ are vectors and b, w are real numbers. We show the detail of the p-stable LSH function in the appendix.

III. Basic Multi-Keyword Fuzzy Search Scheme

A. Main Idea

To design a secure and well functioning search scheme over encrypted data, one has to make three important design choices that are closely inter-related and largely determine the performance of the resulting search scheme, 1). data structure used to build secure indexes and trapdoors; 2). effective search algorithm that can quantify the level of match between keywords in the query and keywords in the index with high efficiency, and 3). security and privacy mechanisms that can be integrated in the above two design choices thus the index privacy and search privacy can be protected.

In this subsection, we outline the key ideas behind our design for 1) and 2). We will present key idea of the data structure and search algorithm in the plaintext format for ease of understanding. More detailed scheme design with integrated security and privacy mechanisms will be described in section III-B.

Our scheme builds index on a per file basis, namely, $\mathcal{I}_\mathcal{D}$ for file $\mathcal{D}$. The index $\mathcal{I}_\mathcal{D}$, containing all the keywords in $\mathcal{D}$, is a m-bit Bloom filter. To support fuzzy and multiple keyword search, we first convert each keyword into a bigram vector and then use LSH functions instead of standard hash functions to insert the keywords into the Bloom filter $\mathcal{I}_\mathcal{D}$. The main steps are illustrated in Fig. 2 and explained as follows.

1) Bigram vector representation of keyword: One key step to build index is the keyword transformation. The LSH function takes a vector as the input and hash "close" vectors to the same value with high probability. We use the following method to transform a string type keyword to its vector representation so that it can be used in the LSH functions. A keyword is first transformed to a bigram set, which contains all the contiguous 2 letters appeared in the keyword. For example, the bigram set of keyword "network" is {ne,et,tw,wo,or,rk}. We use a 26^2-bit long vector to represent a bigram set. Each element in the vector represents one of the 26^2 possible bigrams. The element is set to 1 if the corresponding bigram exists in the bigram set of a given keyword. This bigram vector based keyword representation is not sensitive to the position of misspelling, nor is it sensitive to which letter it was misspelled to. "n*w*twork", "n*v*twork", or "netwo*y*k" will all be mapped to a vector with two-element difference from the original vector. By this representation, a keyword can be misspelled in many different ways but still be represented in a vector that is very close to the correct one, and this closeness (distance) is measured by Euclidean distance, the well-known metric for distance between vector-type data items. This bigram vector representation is robust and inclusive, and key to enabling the use of LSH functions.

2) Bloom filter representation of index/query: Bloom filter has been used to build per document index before [11], [21], for single keyword exact search scenario. Regular hash functions were used that take arbitrary input and hash it to a statistically independent random value as output. With those hash functions, two similar inputs, even if they are only off by

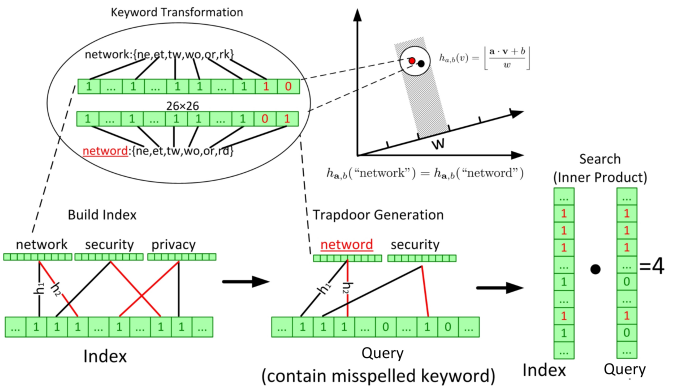

Fig. 2. i).Transform a keyword into a vector. ii).Use two LSH functions h_1, h_2 from the same hash family to generate the index and the query. The word "network" has the same hash value with the misspelled work "netword" under LSH function $h_{\mathbf{a},b}$ because the Euclidean distance between their vector representations is within the pre-defined threshold. iii).The misspelled query matches exactly with the index contains the keywords "network" and "security". (Encryptions are not shown in the example and we use '$\cdots$' to represent all the 0s).

one bit, will be hashed to two totally different random values. Therefore, they can only be used for exact keyword search. In this paper, we adopt a special class of hash functions - locality sensitive hash - to build the index. LSH functions will hash inputs with similarity within certain threshold into the same output with high probability. Fig. 2 shows the idea that a misspelled keyword "netword" in the user query is hashed into the same bucket as the correctly spelled keyword "network" so that a match can be found during the search process. The use of LSH functions in building the per-file Bloom filter based index is the key to implementing fuzzy search. Therefore, indexes and queries now are represented as vectors instead of words. To ease the presentation, we still use the terms *index,query*.

3) Inner product based matching algorithm: As shown in Fig. 2, the final secure index for each file is a Bloom filter that contains all the keywords in the file, where the keywords are first transformed into its bigram vector representation and then inserted into the Bloom filter by LSH functions. The query can be generated in the same way by inserting multiple keywords to be searched into a query Bloom filter. The search can then be done by qualifying the relevance of the query to each file, which in our scheme is done through a simple inner product of the index vector and the query vector. If a document contains the keyword(s) in the query, the corresponding bits in both vectors will be 1 thus the inner product will return a high value. This simple inner product result thus is a good measure of the number of matching keywords.

B. Scheme Construction

We present more detailed description of the proposed scheme in this section, with integrated security and privacy mechanisms. Our scheme is based on symmetric cryptography and consists of four polynomial-time algorithms:

- **KeyGen**(m)**:** Given a security parameter m, output the secret key $SK(M_1, M_2, S)$, where $M_1, M_2 \in \mathbb{R}^{m\times m}$ are invertible matrices and $S \in \{0,1\}^m$ is a vector.
- **Index_Enc**$(SK, \mathcal{I})$: Split the index $\mathcal{I}$ into two vectors $\{\mathcal{I}', \mathcal{I}''\}$ following the rule: for each element $i_j \in \mathcal{I}$, set $i'_j = i''_j = i_j$ if $s_j \in \mathcal{S}$ is 1; otherwise $i'_j = \frac{1}{2}i_j + r, i''_j = \frac{1}{2}i_j - r$ where r is a random number. Then encrypt $\{\mathcal{I}', \mathcal{I}''\}$ with (M_1, M_2) into $\{M_1^T \cdot \mathcal{I}', M_2^T \cdot \mathcal{I}''\}$. Output $Enc_{SK}(\mathcal{I}) = \{M_1^T \cdot \mathcal{I}', M_2^T \cdot \mathcal{I}''\}$ as the secure index.
- **Query_Enc**$(SK, \mathcal{Q})$: Split the query $\mathcal{Q}$ into two vectors $\{\mathcal{Q}', \mathcal{Q}''\}$ following the rule: $q'_j = q''_j = q_j$ if $s_j \in \mathcal{S}$ is 0; otherwise $q'_j = \frac{1}{2}q_j + r', q''_j = \frac{1}{2}q_j - r'$ where r' is another random number. Then encrypt $\{\mathcal{Q}', \mathcal{Q}''\}$ as $\{M_1^{-1} \cdot \mathcal{Q}', M_2^{-1} \cdot \mathcal{Q}''\}$. Output $Enc_{SK}(\mathcal{Q}) = \{M_1^{-1} \cdot \mathcal{Q}', M_2^{-1} \cdot \mathcal{Q}''\}$ as the trapdoor.
- **BuildIndex**$(\mathcal{D}, SK, l)$: Choose l independent LSH functions from the p-stable LSH family $\mathcal{H} = \{h : \{0,1\}^{26^2} \to \{0,1\}^m\}$. Construct a m-bit Bloom filter $\mathcal{I}_\mathcal{D}$ as the index for each file $\mathcal{D}$.
  1) Extract the keywords set $\mathcal{W}_\mathcal{D} = \{w_1, w_2, \cdots\}, w_i \in \{0,1\}^{26^2}$ from $\mathcal{D}$.
  2) For each keyword w_i, insert it into the index $\mathcal{I}_\mathcal{D}$ using $h_j \in \mathcal{H}, 1 \le j \le l$.
  3) Encrypt the index $\mathcal{I}_\mathcal{D}$ using **Index_Enc**$(SK, \mathcal{I}_\mathcal{D})$ and output $Enc_{SK}(\mathcal{I}_\mathcal{D})$.
- **Trapdoor**$(\mathcal{Q}, SK)$**:** Generate a m-bit long Bloom filter for the query $\mathcal{Q}$. For each search keyword q_i, insert q_i using the same l LSH functions $h_j \in \mathcal{H}, 1 \le j \le l$ into the Bloom filter. Encrypt $\mathcal{Q}$ using **Query_Enc**$(SK, \mathcal{Q})$, and output the $Enc_{SK}(\mathcal{Q})$.
- **Search**$(Enc_{SK}(\mathcal{Q}), Enc_{SK}(\mathcal{I}_\mathcal{D}))$**:** Output

$$M_1^T\mathcal{I}' \cdot M_1^{-1}\mathcal{Q}' + M_2^T\mathcal{I}'' \cdot M_2^{-1}\mathcal{Q}''$$

as the search result for the query $\mathcal{Q}$ and the document $\mathcal{D}$,which can be shown as equivalent to compute

$$\mathcal{I}'^T \cdot \mathcal{Q}' + \mathcal{I}''^T \cdot \mathcal{Q}'' = \mathcal{I}^T \cdot \mathcal{Q}$$

Therefore, the inner product of the encrypted index and the trapdoor is equivalent to the inner product of the original index and the query. **Index_Enc**$(SK, \mathcal{I})$, **Query_Enc**$(SK, \mathcal{Q})$ are index and query encryption function respectively, which are tailored for our design from the secure kNN scheme [17].

Discussion: The inner product of the secure index $Enc_{SK}(\mathcal{I}_\mathcal{D})$ and the trapdoor $Enc_{SK}(\mathcal{Q})$ is the exact number of the matching bits in the Bloom filter, which shows whether the query keywords existed in the document. For security consideration, we follow several rules when choosing the secret key SK in *KenGen* and random numbers during the encryption. First, the security parameter m should be long enough to prevent brute forcing attack, e.g., 128 bits. Second, the number of 0s should be approximately equal to the number of 1s in the split vector S in order to maximize the randomness introduced by S. Last, r used during the split process should be picked uniformly from $\mathbb{R}$.

Dataset update A particular advantage of our scheme over the previous multi-keyword search schemes [2], [4] is that our scheme can support dataset updates efficiently, due to the facts

that our scheme doesn't require a pre-defined global dictionary and each document is individually indexed. Therefore, dataset updates, such as *file adding, file deleting* and *file modifying*, can be done efficiently, involving only the indexes of the files to be modified, without affecting any other files.

C. Soundness of the Basic Scheme

The search result of our basic scheme reflects whether a query $\mathcal{Q}$ matches with a file $\mathcal{D}$.

1) Our basic scheme returns the correct results for the exact keyword search. If the query keywords $\mathcal{Q} \subset \mathcal{W}_\mathcal{D}$, the cloud server should include file $\mathcal{D}$ in the result set. Recall that we use the same l hash functions $h_j \in \mathcal{H}, 1 \leq j \leq l$ when building the index and the query. The positions which are set to 1 in the query $\mathcal{Q}$ are also set to 1 the index $\mathcal{I}_\mathcal{D}$, which implicates that the inner product reaches the maximum value that the query can produce. Therefore, the file $\mathcal{D}$ is included in the result set.
2) Our basic scheme returns the correct results with high probability for the fuzzy keyword search. Suppose the keyword $w \in \mathcal{Q}$ is slightly different from the keyword $w' \in \mathcal{W}_\mathcal{D}$, i.e., $d(w, w') \leq r_1$ where r_1 is the distance threshold defined in the LSH function. If $h_j(w) = h_j(w'), h_i \in \mathcal{H}, 1 \leq i \leq l$ for all the l LSH functions, the *Search* returns the maximum value as the exact keyword search.
   If $d(w, w') \leq r_1$ but $h_j(w) \neq h_j(w')$, then we call it a LSH *miss*, which lowers the inner product. If k LSH *misses* happen for w and w' when $d(w, w') \leq r_1$, the probability is $\binom{l}{k}(1-p_1)^k p_1^{l-k}$, where p_1 is the probability defined in the LSH function. Since p_1 is close to 1 in practice, $\binom{l}{k}(1-p_1)^k p_1^{l-k}$ decreases when k increases.
   If $d(w, w') > r_1$, the probability that the LSH functions hash them together is very low. Thus, our basic scheme returns the relatively high inner product with high probability.

False Positive and False Negative A *false positive* is that a keyword w in the index matches with the query keyword q but $d(p, q) > r_2$, and a *false negative* is that a keyword w in the index doesn't match with the query keyword q but $d(p, q) < r_1$, where r_1, r_2 are the parameters defined in the (r_1, r_2, p_1, p_2)-sensitive hash functions.

False Positive: Both the Bloom filter and the LSH functions generate false positive. The false positive rate of a m-bit Bloom filter using l hash functions is $(1 - \frac{1}{m})^{nl}$, where n is the number of the items inserted into the Bloom filter. The false positive caused by LSH is p_2^l, which equals $p(c)$ in p-stable LSH. The false positive rate of our basic scheme is

$$(1 - (1 - p_2)^n (1 - \frac{1}{m})^{n(l-1)})^l \tag{3.2}$$

where n is the number of keywords in the index; m is the length of the Bloom filter and l is the number of LSH functions.

False Negative: Bloom filter will not introduce false negative. Therefore, all the false negatives are generated by the LSH functions. The false negative rate of our basic scheme is:

$$1 - (1 - (1 - p_1)(1 - p_2)^{n-1}(1 - 1/m)^{n(l-1)})^l \tag{3.3}$$

Discussions: Our design returns the documents with the highest inner product score, which works fine when no fuzzy keyword presents in the query. But when there are fuzzy keywords in the query, some of the matched files may be missed from the search result because the inner products may be lowered due to one or more *misses* caused by LSH functions. This increases the false negative rate. One alternative approach is to add documents with relatively high score into the result to reduce the false negative rate. However, the alternative will increase the false positive rate. There is a trade-off between the false positive rate and the false negative rate of our scheme. The data owner can tune the parameters, i.e. m, l, to specifically fit his own accuracy requirements.

D. Security and Privacy analysis

According to [10], the cloud server can build up access patterns by recording the trapdoors and their search results. Therefore, nothing beyond the access pattern and the search result should be leaked under the known ciphertext model.

Theorem 1. *Our basic scheme is secure under the known ciphertext model.*

Before proving the theorem 1, we introduce some notions used in [10].

- *History* is a file set Δ, a index set I built from Δ and a set of queries $W = (w_1, \cdots, w_k)$ submited by users, denoted as $H = (\Delta, I, W_k)$.
- *View* is the encrypted form of a H under some secret key sk, denoted as $V(H)$, i.e., the encrypted files $Enc_{sk}(\Delta)$, the secure indexes $Enc_{sk}(I(\Delta))$ and the trapdoors $Enc_{sk}(W_k)$. Note that the cloud server can only see views.
- *Trace of a history*, denoted as $Tr(H)$, captures the information which can be learned by the cloud server, i.e. the access patterns and the search results induced by H. A trace of the history H is the set of the trace of queries $Tr(H) = \{Tr(w_1), \cdots, Tr(w_k)\}$, and $Tr(w_i) = \{(\delta_j, s_j)_{w_i \subset \delta_j}, 1 \leq j \leq |\Delta|\}$, where s_j is the similarity score between the query w_i and the file δ_j.

Intuitively, given two histories with the same trace, if the cloud server cannot distinguish which of them is generated by the simulator, he cannot learn additional information about the index and the dataset beyond the search result and the access pattern. Now we prove the theorem 1.

Proof: We adopt a similar simulation based proof used in [10]. Denote S is a simulator that can simulate a view V' indistinguishable from a cloud server's view $V(Enc_{sk}(\Delta), I, T(W_k))$. To achieve this, the simulator S does the followings:

- S selects a random $\delta_i' \in \{0,1\}^{|\delta_i|}, \delta_i \in \Delta, 1 \leq i \leq |\Delta|$, and outputs $\Delta' = \{\delta_i', 1 \leq i \leq |\Delta'|\}$.

- S randomly picks two invertible matrices $M_1', M_2' \in \mathbb{R}^{m\times m}$, one split vector $S' \in \{0,1\}^m$ and set $sk' = \{M_1', M_2', S'\}$.
- S constructs the W_k' and the trapdoor $Enc_{sk'}(W_k')$ as follow. For each $w_i \in W_k, 1 \le i \le k$,
  1) Generate a $w_i' \in \{0,1\}^m$. Ensure that the number of 1s in w_i' is same as the number of 1s in w_i but their positions are different. This is easy to achieve because there are at most l 1s in w_i and $l \ll m$. Output $W' = \{w_i', 1 \le i \le k\}$.
  2) Generate the trapdoor for each $w_i' \in W'$, i.e., $Enc_{sk'}(w_i')$ for $1 \le i \le k$. Then S sets $Enc_{sk'}(W_k') = \{Enc_{sk'}(w_1'), \cdots, Enc_{sk'}(w_k')\}$.
- To generate $I(\Delta')$, S first generate a m-bit null vector for each $\delta_i' \in \Delta', 1 \le i \le |\Delta'|$ as the index, denoted as $I_{\delta_i'}$. Then S does the following:
  1) For each $w_i \in W_k$, if $w_i \subset \delta_j, 1 \le j \le |\Delta|$, S sets $I_{\delta_j'}$ as $I_{\delta_j'} + w_i'$.
  2) S converts each $I_{\delta_j'}, 1 \le j \le |\Delta'|$ into a vector in $\{0,1\}^m$ by replacing the elements bigger than 1 with 1.
  3) S generates $Enc_{sk'}(I(\Delta'))$ as $Enc_{sk'}(\{I_{\delta_j'}\}, 1 \le j \le |\Delta'|)$.
- S outputs the view $V' = (\Delta', Enc_{sk'}(I(\Delta')), Enc_{sk'}(W_k'))$.

The correctness of the construction is easy to demonstrate. The secure index $Enc_{sk'}(I(\Delta'))$ and the trapdoor $Enc_{sk'}(W_k')$ generate the same trace as the one that the cloud server has. We claim that no probabilistic polynomial-time (P.P.T.) adversary can distinguish between the view V' and $V(H)$. Particularly, due to the semantic security of the symmetric encryption, no P.P.T adversary can distinguish between $Enc_{sk}(\Delta), \Delta'$. And the indistinguishability of indexes and trapdoors is based on the indistinguishability of the secure kNN encryption and the random number introduced in the split processes. ■

While our basic scheme is secure under the *known ciphertext model*, it is vulnerable under *known background model*. Under *known background model*, the cloud server may obtain the background information, i.e., the keyword frequency and distribution among the dataset [2]. And these information can be used to infer some keyword w to its trapdoor $Enc_{sk}(w)$. Because the secure kNN method is vulnerable to linear analysis [22], cloud server can launch the linear analysis using the keyword and its trapdoor pair if the number of keyword-trapdoor pairs is large enough. So the index might be partially or entirely recovered. To address the potential privacy violation problem under known background attack, we propose an enhanced scheme to strengthen our basic scheme.

IV. Enhanced Multiple Keywords Fuzzy Search Scheme

In this section, we present our enhanced scheme, which improves the security under $known\ background\ model$.

A. Enhanced Scheme

As showed in section 3, under *known background model*, the adversary potentially can recover the encrypted indexes through linear analysis and further infer the keywords in the index. To secure the linkage between the keywords and the Bloom filter, we introduce an extra security layer, i.e., a pseudo-random function f.

Our enhanced scheme contains the following four processes:

- **KeyGen**(m, s)**:** Given a parameter m, generate the secret key $SK(M_1, M_2, S)$, where $M_1, M_2 \in \mathbb{R}^{m\times m}$ are invertible matrices while $S \in \{0,1\}^m$ is a vector. Given another parameter s, generate the hash key pool $HK = \{k_i | k_i \xleftarrow{R} \{0,1\}^s, 1 \le i \le l\}$.
- **BuildIndex**$(\mathcal{D}, SK, l)$: Choose l independent LSH functions from the p-stable LSH family $\mathcal{H}$ and one pseudo-random function $f : \{0,1\}^* \times \{0,1\}^s \to \{0,1\}^*$. For each file $\mathcal{D}$,
  1) Extract the keywords set $\mathcal{W} = \{w_1, w_2, \cdots\}$ from $\mathcal{D}$.
  2) Generate a m-bit Bloom filter $\mathcal{I}_\mathcal{D}$. Insert $\mathcal{W}$ into $\mathcal{I}_\mathcal{D}$ using the hash functions $\{g_i | g_i = f_{k_i} \circ h_i, h_i \in \mathcal{H}, 1 \le i \le l\}$.
  3) Encrypt the $\mathcal{I}_\mathcal{D}$ with SK and return $Enc_{SK}(\mathcal{I}_\mathcal{D})$ as the index.
- **Trapdoor**$(\mathcal{Q}, SK)$**:** Generate a m-bit long Bloom filter. Insert the $\mathcal{Q}$ using the same hash functions g_i, i.e., $g_i = f_{k_i} \circ h_i, h_i \in \mathcal{H}, 1 \le i \le l$ into the Bloom filter. Encrypt the $\mathcal{Q}$ with SK and return the $Enc_{SK}(\mathcal{Q})$ as the trapdoor.
- **Search**$(Enc_{SK}(\mathcal{Q}), Enc_{SK}(\mathcal{I}_\mathcal{D}))$**:** Output the inner product $< Enc_{SK}(\mathcal{Q}), Enc_{SK}(\mathcal{I}_\mathcal{D}) >$ as the search result for the query $\mathcal{Q}$ and the document $\mathcal{D}$.

Note that the extra security layer in the enhanced scheme doesn't affect the search result because pseudo-random functions are collision free. However, in practice, we can use HMAC-SHA1, since the collision rate of HMAC-SHA1 is very low.

B. Security Analysis of Enhanced Scheme

Under *known background model*, we assume the cloud server obtains not only the trace of a history, but also a certain number of the keyword and trapdoor pairs, denoted as (w_i, T_i). Intuitively, the adversary with the keyword and trapdoor pairs should not be able to distinguish the view generated by the simulator from the view he owns. We use a simulation based approach to prove the security of our enhanced scheme.

Theorem 2. *Our enhanced scheme is secure under the known background model.*

Proof: Denote S is a simulator that can simulate a view V' indistinguishable from the view the cloud server has. Then we construct the simulator as follows:

- To generate Δ', S selects a random $\delta_i' \in \{0,1\}^{|\delta_i|}, \delta_i \in \Delta, 1 \le i \le |\Delta|$, and outputs $\Delta' = \{\delta_i', 1 \le i \le |\Delta'|\}$.

- S randomly picks two invertible matrices $M_1', M_2' \in \mathbb{R}^{m\times m}$, one split vector $S' \in \{0,1\}^m$ and a pseudo-random permutations F Set $sk' = \{M_1', M_2', S'\}$.
- S constructs the query W_k' and the corresponding trapdoor as follow. For each $w_i \in W_k, 1 \le i \le k$,
  1) Generate a $w_i' \in \{0,1\}^m$. Ensure that the number of 1s in w_i' is same as the number of 1s in w_i but their positions are different. This is easy to achieve because there are at most l 1s in w_i and $l \ll m$.
  2) Generate the trapdoor for each $w_i' \in W'$, i.e., $Enc_{sk'}(w_i') = Enc_{sk'}(F(w_i')), 1 \le i \le k$. Then S sets $Enc_{sk'}(W_k') = \{Enc_{sk'}(F(w_1')), \cdots, Enc_{sk'}(F(w_k'))\}$.
- To generate $I(\Delta')$, S first generate a m-bit null vector for each $\delta_i' \in \Delta', 1 \le i \le |\Delta'|$ as the index, denoted as $I_{\delta_i'}'$, then does the following:
  1) For each $w_i \in W_k$, if $w_i \subset \delta_j, 1 \le j \le |\Delta|$, S sets $I_{\delta_j'}$ as $I_{\delta_j'} + w_i'$.
  2) S converts each $I_{\delta_j'}, 1 \le j \le |\Delta|$ into a vector in $\{0,1\}^m$ by replacing the elements bigger than 1 with 1.
  3) $I(\Delta') = Enc_{sk'}(\{F(I_{\delta_j'})\}, 1 \le j \le |\Delta|)$.
- S outputs the view $V' = (\Delta', I(\Delta'), Enc_{sk'}(W_k'))$.

The construction is correct since the search result on $I(\Delta')$ with the trapdoor $Enc_{sk'}(W_k')$ is same as the trace which the cloud server has. We claim that no P.P.T. adversary with the keyword and trapdoor pairs can distinguish the view V' from $V(H)$. Particularly, due to the semantic security of the symmetric encryption, no P.P.T adversary can distinguish $Enc_{sk}(\Delta)$ from Δ'. And the P.P.T adversary with the keyword and trapdoor pairs cannot distinguish the output of the linear analysis from a random string because of the indistinguishability of the pseudo-random function F. ■

V. Experimental Results

We use the recent 10 years' IEEE INFOCOM publication as our experiment dataset which contains more than 3600 files. We extract 5734 keywords in total, and the average number of the keywords in a paper is 147 while the minimum and the maximum are 112 and 175 respectively. We use a 2-stable $(\sqrt{3}, 2, p_1, p_2)$-LSH family to build the index which supports 1 edit distance difference, where $p_1 = P(\sqrt{3}) = 0.558864, p_2 = P(3/4) = 0.285932$ using $P(x) = -\frac{\left(1-e^{-\frac{x^2}{2}}\right)\sqrt{\frac{2}{\pi}}}{x} + \text{Erf}\left[\frac{x}{\sqrt{2}}\right]$. We choose $k = 10, l = 30$, and set $n = 200, m = 8000$. We implement our schemes on a desktop PC equipped with Intel Core i3 processor at 3.3 GHz and 4 Gb RAM, which has the same computation power with an Amzaon EC2 M1 Medium instance. To generate a fuzzy keyword in a query, we randomly choose one letter from a keyword and replace it with another letter. We allow at most two fuzzy keywords in a query.

A. Efficiency

1) Index and Trapdoor Generation: The index generation process is a one-time computation which contains two major steps: the Bloom filter generation and the encryption. During the Bloom filter generation, the computation mainly comes from the hash function calculation. Figure 3.(a) shows the Bloom filter generation time for the index and the trapdoor Bloom filter. The generation time increases linearly respect to the number of the inserted keywords. The trapdoor generation time is very close to the index generation time due to the identical procedure. The encryption time which involves the matrix multiplications is showed in figure 3.(b). The time cost of encryption increases linearly respect to the number of the files in the dataset.

2) Search over Encrypted Index: The search operation executed at the cloud server side consists of computing the inner product calculation for all the files in the dataset. Figure 3.(c) shows the search time grows linearly with the size of the file set while the number of keywords in the query has little impact as showed in figure 3.(d). This is intuitive because the search process needs to go over all the files in the dataset before the cloud server can get the final result. The inner product computation is only related to the length of the index, so the computation time changes little in figure 3.(d).

B. Result Accuracy

We adopt the definitions of the widely used performance metrics, precision and recall to measure the search result accuracy. Denote tp as true positive, fp as false positive and fn as false negative, then the precision equals to $\frac{tp}{tp+fp}$ while the recall is $\frac{tp}{tp+fn}$. To generate the fuzzy queries, we randomly pick two keywords and modify them into the fuzzy keywords.

Figure 4.(a) shows the performance metrics of our scheme according to k. Note that there is no recall for the exact matching because the false negative doesn't exist. One observation is that precision is very low when k is small, i.e. 5% at $k = 1$. Because that multiple LSH functions are used together to enlarge the gap between p_1 and p_2. So when the k is small, the gap is not big enough to distinguish the different keywords, and most of the files in the dataset have been returned, which leads to high fp and low fn. The jump at $k = 5$ is due to the gap between p_1 and p_2 increases exponentially respect to k. After a certain k, i.e., $k = 8$, the precision is remained at a high level, which is above 90% for the exact search and above 80% for the fuzzy search. Another observation is that the recall drops when increasing the k. This is because that increasing the k will cause more false negatives. In general, the false positive and the false negative cannot be improved at the same time.

Another important parameter is the number of the keywords in the query. Figure 4.(b) shows the precision of the exact match decreasing slightly, from 100% to 96% while the number of the keywords in the query increases from 1 to 10. This is reasonable because the false positive generated by each keyword accumulates. But the precision for the fuzzy search doesn't show the same pattern. It is slightly increased from 70% to 81% when the number of the keywords in the query increases from 1 to 10. The reason is that the false positive caused by the LSH functions contributes much more

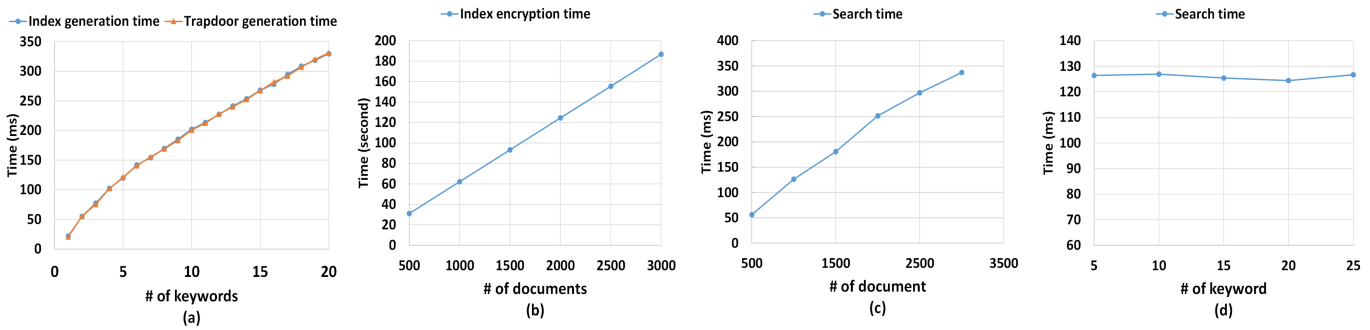

Fig. 3. (a) The Bloom filter generation time for a single file v.s. # of the keywords; (b) The encryption time for all the indexes v.s. the dataset size. The computation time for the *search* process; (c) Different size of the file set with the fixed keywords number k=5 in the query; (d) Different # of the keywords in the query with the fixed file size 1000

than the false positive introduced by the Bloom filter. As the portion of the fuzzy keywords decreases, the impact of the false positive caused by the fuzzy keyword is reduced since the fuzzy keywords contribute less in the search result.

VI. Related Works

A. Searchable Encryption Scheme without Fuzzy Search Support

1) Single Keyword Searchable Encryption: Song et al. [1] studied this problem first under the symmetric key setting for email systems. Their scheme didn't contain an index, thus, the search operation went through the entire file. Goh proposed a secure index using the Bloom filter in [11]. Curtmola et al. gave the formal definition of the searchable encryption and proposed an index scheme based on the inverted list in [10]. In [8], Wang et al. solved the result ranking problem utilizing the keyword frequency and order-preserving encryption. Boneh et al. [3]proposed the first searchable encryption scheme using the asymmetric encryption scheme. All of these works only supported the single keyword search over the encrypted data.

2) Multiple Keywords Searchable Encryption: To enrich the search functionality, the schemes supporting conjunctive keywords search have been proposed [4]–[7], [23]–[25]. Many works which supported the conjunctive keyword search, subset search, range queries were using the asymetric encryption [5]–[7]. [23]–[25] used the predicate encryption to achieve the conjunctive keywords search over encrypted data. In [26], a logarithmic-time search scheme was presented to support the range queries. Cao et al. [4] proposed a privacy-preserving multi-keyword ranked search scheme using symmetric encryption. Sun et al. [2] proposed an efficient privacy-preserving multi-keyword supporting cosine similarity measurement. However, none of the schemes can support fuzzy keyword search.

B. Searchable Encryption Scheme support Fuzzy Search

Li et al. proposed a wildcard based fuzzy search over encrypted data in [12]. Then Liu et al. [13] improved the scheme by reducing the index size. In [27], the LSH functions are used to generate file index. But it took two rounds of communication to achieve results ranking and only supported the single keyword search. All the aforementioned schemes only support the single keyword search, the fuzzy match OR the exact match. In [14], Chuah et al. improved [12] by introducing a tree structure index and enriched the search functionality by treating the pre-defined phrases, for example, "cloud computing", as a single keyword.

VII. Conclusion

In this paper, we tackled the challenging multi-keyword fuzzy search problem over the encrypted data. We proposed and integrated several innovative designs to solve the multiple keywords search and the fuzzy search problems simultaneously with high efficiency. Our approach of leveraging LSH functions in the Bloom filter to construct the file index is novel and provides an efficient solution to the secure fuzzy search of multiple keywords. In addition, the Euclidean distance is adopted to capture the similarity between the keywords and the secure inner product computation is used to calculate the similarity score so as to enable result ranking. We proposed a basic scheme as well as an improved scheme in order to meet different security requirements. Thorough theoretical security analysis and experimental evaluation using real-world dataset were carried out to demonstrate the suitability of our proposed scheme for the practice usage.

Acknowledgment

This work was supported in part by US National Science Foundation under grants CNS-1217889 and CNS-1338102.

References

[1] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," *S&P 2000*, vol. 8, pp. 44–55, 2000.

[2] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *ASIACCS 2013*, May 2013.

[3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," *EUROCRYPTO 2004*, pp. 506–522, 2004.

[4] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *INFOCOM 2011*, pp. 829–837, 2011.

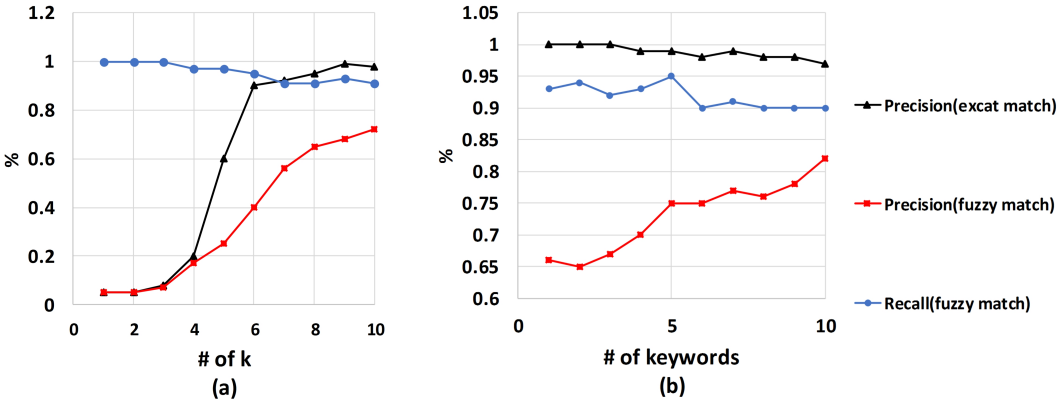

Fig. 4. (a). The performance matrices of varying the value of k. We fix the query size to 5; (b).The performance matrices of varying # of keywords in the query. We fix the k=10.

[5] Y. Hwang and P. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," *Pairing 2007*, pp. 2–22, 2007.
[6] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," *Theory of Cryptography*, vol. 4392, pp. 535–554, 2007.
[7] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," *ACNS 2004*, vol. 3089, pp. 31–45, 2004.
[8] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," *ICDCS 2010*, pp. 253–262, 2010.
[9] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," *ACNS 2005*, vol. 3531, pp. 442–455, 2005.
[10] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *CCS 2006*, vol. 19, pp. 79–88, 2006.
[11] E.-J. Goh, "Secure indexes," *Cryptology ePrint Archive on October 7th*, pp. 1–18, 2003.
[12] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *IEEE INFOCOM 2010, mini-conference*, San Diego, CA, USA, March 2010.
[13] C. Liu, L. Zhu, L. Li, and Y. Tan, "Fuzzy keyword search on encrypted cloud storage data with small index," *ICCCIS 2011*, pp. 269–273, 2011.
[14] M. Chuah and W. Hu, "Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data," *ICDCSW 2011*, pp. 273–281, 2011.
[15] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," *Proceedings of the 30th ACM symposium on Theory of computing*, vol. 126, pp. 604–613, 1998.
[16] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, pp. 422–426, 1970.
[17] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," *SIGMOD 2009*, pp. 139–152, 2009.
[18] M. Li, S. Yu, K. Ren, and W. Lou, "Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings," in *SecureComm 2010*, Singapore, September 7-9 2010.
[19] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *IEEE INFOCOM 2010*, San Diego, CA, USA, March 2010.
[20] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," *SCG 2004*, 2004.
[21] S. M. Bellovin and W. R. Cheswick, "Privacy-enhanced searches using encrypted bloom filters," 2007.
[22] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," 2013.
[23] E. Shi, J. Bethencourt, T. h. Hubert, C. Dawn, and S. A. Perrig, "Multi-dimension range query over encrypted data," *S&P 2007*, pp. 350–364, 2007.
[24] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," *TCC 2009*, pp. 457–473, 2009.
[25] N. Attrapadung and B. Libert, "Functional encryption for inner product: Achieving constant-size ciphertexts with adaptive security or support for negation," *PKC 2010*, vol. 6056, pp. 384–402, 2010.
[26] Y. Lu, "Privacy-preserving logarithmic-time search on encrypted data in cloud," *NDSS 2012*, 2012.
[27] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," *28th International Conference on Data Engineering*, pp. 1156–1167, 2012.

APPENDIX

Definition 1. *A distribution $\mathcal{D}$ over $\mathbb{R}$ is called a p-stable distribution, if $\exists p \geq 0$ such that for n real numbers $v_1, v_2, \cdots, v_n$ and i.i.d. variables $X_1, X_2, \cdots, X_n$ with same distribution, the summation $\sum_i v_i X_i$ also follows the distribution $\mathcal{D}$ with the variable $(\sum_i |v_i|^p)^{1/p} X$, where X is a random variable with distribution $\mathcal{D}$.*

- when $p = 1$, it is **Cauchy distribution**, defined by the density function $f_p(x) = \frac{1}{\pi}\frac{1}{1+x^2}$, is 1-stable.
- when $p = 2$, it is **Gaussian distribution**, defined by the density function $f_p(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable.

The p-stable LSH function is given by:

$$h_{a,b}(v) = \left\lfloor \frac{\mathbf{a} \cdot \mathbf{v} + b}{w} \right\rfloor$$

where $\mathbf{a}$ is a d-dimensional vector, $b \in [0, w]$ is a real number and w is a fixed constant for one family.

The hash function $h_{\mathbf{a},b}(\mathbf{v}) : \mathbb{R}^d \to \mathbb{Z}$ maps a d-dimensional vector $\mathbf{v}$ onto the set of integers. By choosing different $\mathbf{a}, b$, different hash functions in the family can be generated. Given two vectors $\mathbf{v}_1, \mathbf{v}_2$, let $c = \| \mathbf{v}_1 - \mathbf{v}_2 \|_p$, it is easy to see

$$p(c) = Pr_{\mathbf{a},b}[h_{\mathbf{a},b}(\mathbf{v}_1) = h_{\mathbf{a},b}(\mathbf{v}_2)] = \int_0^w \frac{1}{c} f_p(\frac{t}{c})(1 - \frac{t}{w})dt,$$

where f_p is the probability density function. Denote $\epsilon = r_2/r_1$, then p-stable LSH is showed $(r, \epsilon r, p_1, p_2)$-sensitive where $p_1 = p(r), p_2 = p(\epsilon r)$ in [20].