Privacy-aware BedTree Based Solution for Fuzzy Multi-keyword Search over Encrypted Data

M. Chuah, W. Hu

Department of Computer Science & Engineering
Lehigh University
Bethlehem, PA 18015
chuah@lehigh.edu, weh210@lehigh.edu

Abstract—**As Cloud Computing technology becomes more mature, many organizations and individuals are interested in storing more sensitive data e.g. personal health records, customers related information in the cloud. Such sensitive data needs to be encrypted before it is outsourced to the cloud. Typically, the cloud servers also need to support a keyword search feature for these encrypted files. Traditional searchable encryption schemes typically only support exact keyword matches. However, users sometimes have typos or use slightly different formats e.g. "data-mining" versus "data mining". Thus, fuzzy keyword search is a useful feature to have. Recently, some researchers propose using wildcard based approach to provide fuzzy keyword search. They also propose a solution for multi-keyword search. Their approaches have some limitations, namely (a) their fuzzy keyword search solution consumes large storage size since it inserts every fuzzy keyword as a leaf node in the index tree, (b) their fuzzy single-keyword search solution does not support multi-keyword search, (c) the existing multi-keyword search scheme does not provide efficient incremental updates. In this paper, we propose a privacy-aware bedtree based approach to support fuzzy multi-keyword feature. Incremental updates can be easily done using our solution. We have implemented our solution. Our evaluation results show that our approach is more cost-effective in terms of storage size and construction time. Our search time is usually better than the wildcard approach for multi-keyword queries where many encrypted files are returned using single-word queries for approaches that do not support multi-keyword queries.**

Keywords-multi-keyword search, encrypted cloud data, fuzzy keyword search, co-occurrence probability

I. INTRODUCTION

As cloud computing technology becomes more mature, organizations and individuals wish to store more sensitive data e.g. personal health records, emails, customer related documents in the cloud. By storing such data in the cloud, the data owners do not have to worry about data storage and maintenance. Furthermore, they can enjoy the increased data availability provided by the cloud servers. However, data owners may not trust the cloud servers. Hence, data owners often encrypt the outsourced data for data privacy and mitigation against unauthorized accesses. Unfortunately, data encryption, if not done appropriately, may reduce the effectiveness of data utilization. In addition, data owners often want to share their published data with many users, some of whom they may not anticipate e.g. new subscribers when the data is outsourced. Furthermore, each user may be interested in only some but not all of the data files published by a data publisher. Typically, a user retrieves files of interest to him/her via keyword search instead of retrieving all the encrypted files back which may not be feasible in cloud computing scenarios. Such keyword-based search technique has been widely used in our daily life e.g. Google plaintext keyword search [1]. Thus, any data encryption solution that is used for encrypting data files or meta-data descriptions (e.g. keywords) of these files should not restrict a user's ability to perform keyword search. In addition, the data encryption solution should provide keyword privacy since keywords usually contain important information about the data files e.g. customers' credit card numbers. Simple encryption of keywords can protect keyword privacy but it may make it hard for users to perform keyword search.

In recent years, searchable encryption techniques [2-7] have been developed to allow users to securely search over encrypted data. These techniques typically build an index for each keyword, and associate that index with the files containing that keyword. Using trapdoors of keywords within the index structure preserve both file content and keyword privacy while allowing data users to conduct keyword searches. However, these existing techniques are overly restrictive because they only allow exact keyword search. Often, users that perform searches have typos in their input strings e.g. typing "wireiess" instead of "wireless", or the data format may not be the same e.g. "data-mining" versus "data mining". A spell-check mechanism can be used. However, this does not completely solve the problem since it may require additional user interactions to determine the correct word from a list of candidates suggested by the spell-check algorithm. Besides, the spell check algorithm may not be smart enough to differentiate between two actual valid words. In [9], the authors exploit edit distance to quantify keywords similarity and designed two advanced techniques (wildcard-based and gram-based techniques) to construct storage-efficient fuzzy keyword sets. However, their solution can only provide a single fuzzy keyword search e.g. searching for all data files that contain any keyword with an edit distance of 1 from the word "network". A user that is looking for data files containing the words "ad hoc network" will have to conduct three searches with keywords "ad", "hoc" and "network". Then, he will have to decrypt the meta-descriptions of all the three returned lists and do an intersection to extract the relevant list that contains all three words "ad hoc network". This process consumes communication bandwidth as well as additional decryption time on the client side to determine a relevant list of encrypted files from multiple returned lists. In summary, the solution in [9] lacks the multi-keyword search feature. In [10], the authors propose a solution for multi-keyword ranked search. However, their solution is not efficient for incremental updates because their solution requires matrices to be built a priori based on a set of keywords present in the whole collection.

In this work, we propose a privacy-aware bedtree based solution that supports fuzzy multi-keyword search and incremental updates. Given a collection of files, our solution first constructs a list of useful keywords (including multi-keywords). We currently use co-occurrence probability approach to identify useful multi-keywords for our published data items. More sophisticated technique can be added in the future. Next, we construct all relevant fuzzy keyword sets with the appropriate edit distances that we wish to support. Bloom filters that incorporate all the words in the fuzzy keyword sets of various edit distances are constructed for every keyword. Then, we construct the index tree for the whole data file collection where each leaf node consists of a hash value of a keyword, one or two data vectors that represent the n-gram of that keyword, and several bloom filters (one for each edit-distance value). Any new information related to additional files can be easily inserted into the existing index tree. Later, we submit both the collection of encrypted data files and the index tree to the cloud server. A user submits a query that consists of the hash value of the keyword, the desired edit distance, and a list of hash values of the words included in the associated fuzzy keyword sets to the cloud server. The cloud server will then search the index tree to retrieve relevant keywords and their associated list of encrypted file identifiers which identify files containing these keywords. Our evaluation results based on two publication datasets (both IEEE and ACM publications) show that this proposed solution is effective in terms of storage and construction cost. The searching time for a multi-keyword query is better than what is proposed in [9] if each single-keyword associated with that multi-keyword search returns many encrypted file identifiers.

The rest of the paper is organized as follows: Section II summarizes the features of relevant related work. Section III describes the system model, the threat model, and our design goals. We also provide some definitions and necessary background for the techniques that we use in this paper. Section IV describes how privacy-aware bedtree index tree is constructed. Section V describes how we support fuzzy multi-keyword search. Section VI presents the evaluation results that we have performed. We conclude with some discussions of the near future work that we intend to carry out in Section VII.

II. RELATED WORK

A. *Plaintext Fuzzy Keyword Search*

Plaintext fuzzy keyword search solutions have been proposed [15-17]. These solutions are based on approximate string matching techniques. In [15], the authors propose some efficient merging algorithms to merge inverted lists of grams generated from stored strings so as to obtain approximately matched strings faster. Various indexing techniques such as metric trees, suffix trees, and q-gram methods are used. In [16], the authors propose techniques to reduce the storage cost of the gram-based index tree constructed for approximate string matching. In [17], the authors further enhance the technique to support interactive fuzzy plaintext string matching. However, such approaches cannot be directly used for searchable encryption because they can be easily attacked using dictionary and statistics attacks.

B. *Searchable Encryption & Fuzzy Keyword Search*

Searchable encryption using traditional cryptography approaches have been discussed in [2-7]. Song et al [3] propose the first construction of searchable encryption. In their approach, each word in the document is encrypted independently using a special two-layered encryption construction [9]. Boneh et al [5] proposed a public key based searchable encryption scheme where a user A can send a key to a server to allow the server to search data items that are encrypted using A's public key for the presence of certain keywords. Goh [4] proposed an efficient secure index construction called Z-IDX which is built using pseudo-random functions and Bloom filters. A Bloom filter containing trapdoors of all unique words is constructed for each file, and stored on the server. To search for a word, the user sends the trapdoor of the word to the server. The server checks if any Bloom filter contains the trapdoor of the query word, and returns the corresponding file identifiers. Curmola et al [7] and Chang et al [18] both proposed similar index approaches where a single encrypted hash table index is built for the whole file collection. In the index table, each entry consists of the trapdoor for a keyword and an encrypted set of file identifiers whose corresponding data files contain the keyword. The authors in [7] provide tighter definitions of security guarantees and solutions that can address stronger attacks where malicious users can adapt their queries based on outcomes of previous queries. The authors in [8] and [19] proposed solutions for conducting conjunctive keyword search, subset query, and range query over encrypted data. However, all these techniques support only exact keyword search.

In [9], the authors propose a scheme that allows efficient fuzzy keyword search over encrypted data. However, the storage requirement of their scheme grows as the collection of encrypted files increases because they store all fuzzy keywords as indices for searching purposes. In addition, their paper does not discuss semantic search that is based on multi-keyword e.g. searching for all files that are related to "information retrieval" needs to be translated into two queries, one for the word "information" and one for the word "retrieval". The user then needs to decrypt the returned lists for the two queries to determine which are the relevant ones (assuming that the returned lists contain some meta-data descriptions of the contents of the encrypted files). In the worst case when such meta-data descriptions are not available, the user needs to ask for all the encrypted files from the two returned responses, decrypt them before he can select the relevant ones. Such an approach is both time consuming and inefficient (in terms of communication bandwidth and CPU consumption on the client side). In [10], the authors propose a solution for supporting the multi-keyword ranked search over encrypted data. Their solution requires prior knowledge of all keywords for a collection of data files so that some two dimensional matrices can be constructed. This means that the matrices need to be re-constructed when new files are made available and hence is not efficient for incremental updates.

III. Problem Formulation

A. System Model

As in [9], we consider a cloud data hosting service with three entities, namely data owner, cloud server and data users as shown in Fig 1. Data owner has a collection of data files *F* to be outsourced to a cloud server. All files in *F* are encrypted and form a collection of encrypted files, *C*. To provide the searching capability over *C*, the data owner also builds an encrypted searchable index *I* from *F*, and then outsource both the index *I* and the collection of encrypted files, *C* to the cloud server. To search the document collection for certain keywords, an authorized user constructs corresponding trapdoors for these keywords, *T,* through some search control mechanisms. Upon receiving the trapdoors *T* from data users, the cloud server will search the index *I* and return the relevant set of encrypted documents. For fuzzy keyword search, the cloud server returns the search results according to the following rules: (a) if the user's searching input exactly matches a pre-stored keyword, the server will return the files containing that keyword, (b) if there exist typos and/or format differences in the searching input, then the server will return the closest possible results based on some pre-specified similarity measures (will be formally described in Section III.D).

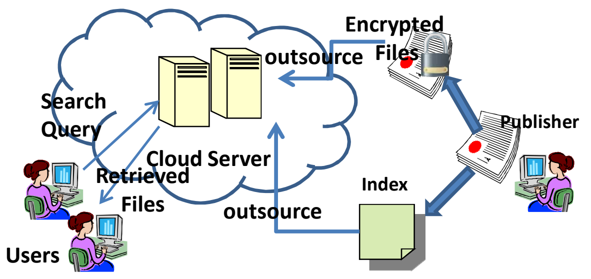

Fig 1: System Architecture for Publishing/Querying Encrypted Files

B. Threat Model

In our work, we use the same threat model described in [9]. We assume that cloud servers are "honest-but-curious". Specifically, a cloud server will not delete encrypted data files or index from its storage. It will also correctly follow the designated protocol specification. However, it is curious to analyze data in its storage and message flows so as to learn additional information. Based on what information a cloud server knows, any design should address the following two threat models described in [10]:

- In the Known Ciphertext Model, a cloud server is supposed to only know encrypted dataset *C* and searchable index *I.*
- In the stronger Known Background Model, a cloud server is assumed to have additional facts on the dataset e.g. the subject and its related statistical information, besides what can be accessed in the known ciphertext model. For instance, a cloud server can utilize the document frequency or keyword frequency [11] to infer keywords in the query.

C. Design Goals

To support fuzzy multi-keyword search for outsourced cloud data using the above system and threat models, our system design should achieve the following security and performance guarantees:

- Fuzzy Multi-keyword search: our search scheme should support fuzzy multi-keyword queries and return a list of encrypted file identifies that contain relevant keywords.
- Privacy preserving: our solution should meet the privacy requirement described in Section III.B and prevent a cloud server from learning additional information from any dataset and its associated index.
- Efficiency: The above goals on functionality and privacy should be accomplished with low communication, storage, and computation overhead.

D. Preliminaries

Before we explain how we can construct an index tree based on gram-counting order using the technique described in [12], we first give a few definitions. Then, we describe the properties that a string order should possess for constructing the gram-counting order index tree used in our design. The definitions given here are summarized from [9] and [12].

Definition 1 (**Edit Distance**) Given two strings s_i and s_j, the edit distance between s_i and s_j is defined as the minimum number of primitive operations needed to transform s_i to s_j, denoted by $d(s_i, s_j)$.

Definition 2 (**String Order**) Given the string domain $\sum^*$, a string order is a mapping function $\phi : \sum^* \rightarrow N$, mapping each string to an integer value. The mapping function is chosen such that it satisfies the following properties:

Property 1: It takes linear time to verify if $\phi(s_i)$ is larger than $\phi(s_j)$ for any string pair s_i and s_j.

Property 2: It is lower bounding i.e. it efficiently returns the minimal edit distance between string q and any $s_l \in (s_i, s_j)$.

Property 3: Given two string intervals $[s_1^l, s_1^u]$ and $[s_2^l, s_2^u]$, the function ϕ is pairwise lower bounding i.e. it returns the lower bound on the distance between any $s_i \in (s_1^l, s_1^u)$ and $s_j \in (s_2^l, s_2^u)$

Property 4: Given any string interval $[s_i, s_j]$, the function ϕ is length bounding i..e. it efficiently returns an upper bound on the length of any string $s_l \in (s_i, s_j)$.

Definition 3 (**Fuzzy Keyword Search**) Given a collection of N encrypted files $C=(F_1, F_2, \ldots F_N)$ stored in a cloud server, a set of distinct keywords $W=\{w_1, w_2, .. w_p\}$ with predefined edit distance *d*, and a searching input *(f(w),k)*(*where f(w)* is a one way function that provides keyword privacy) and k is the desired edit distance *(k<=d)*, the execution of a fuzzy keyword search returns a set of file identifiers whose corresponding data files possibly containing the word *w*, denoted as FID_w: *if* $w= u$, $u \varepsilon W$, then return FID_u; otherwise, if $w \notin W$, then return $\{FID_u\}$ where *ed(w,u) <=k*, assuming that *k<=d*.

Definition 4 (**Trapdoors of keywords**) Trapdoors of the keywords are realized by applying a one-way function f as follows: Given a keyword *w,* and a secret key *sk,* we compute the trapdoor of *w* as $T_w = f(sk, w)$. The trapdoors of keywords help us to achieve query and index privacy.

IV. Privacy-Aware BedTree Construction

Next, we summarize how an index tree for keywords can be built using the gram counting order [12]. An n-gram is a contiguous sequence of n characters from a string *s.* Given string *s,* there exists $|s|+n-1$ overlapping n-grams. For example, the string s_1=*"network"* for n=2, the n-gram set $Q(s_1)=\{\#n,ne,\ et,\ tw,wo,rk,\ k\#\}$. An n-gram set can be represented as a vector in a high dimensional space where each dimension corresponds to a distinct n-gram. To compress the information on the vector space, one can use a hash function to map each-n-gram to a set of L buckets (L=4 is shown in Fig 2), and count the number of n-grams in each bucket. Thus, the n-gram set is transformed into a vector of L non-negative integers. For example, in Fig 2, after hashing the seven 2-grams of the string s_1 to four buckets, we get a 4-dimensional vector $v_1=<2,2,3,1>$ in the gram-counting space. If v_i and v_j are the L-dimensional bucket vector representations of s_i and s_j respectively, then the authors in [12] show that the edit distance between s_i and s_j is no smaller than

$$\max\left(\sum_{v_i[l]>v_j[l]} \frac{v_i[l]-v_j[l]}{n}, \qquad \sum_{v_j[l]>v_i[l]} \frac{v_i[l]-v_j[l]}{n}\right) \qquad \text{for}$$

$1 \le l \le L$. (Eqn 1) [12].

Assume that for a given string interval $[s_i,s_j]$, the lower and upper bound values of bucket *m* are *lb[m]* and *ub[m]* (for $1 \le m \le L$). After transforming the query string *q* to a vector v_q in gram counting order, we can apply Eqn(1) with *lb[m] and ub[m]* to compute some new lower bound on the edit distance from *q* to any string contained in the interval. That allows us to see if we need to search through that string interval for strings that have an edit distance below a certain threshold *d* from the query string *q*.

Using the bedtree construction procedure described in [12], one can construct a bedtree-based index tree with three levels as shown in Fig 3 where each leaf node contains information about the data vector of a word *w (denoted as* v_w*)*, the word *w,* and a list of encrypted file identifiers that contain the word *w.* Each intermediate node contains information that summarizes the string interval of its children leaf nodes.

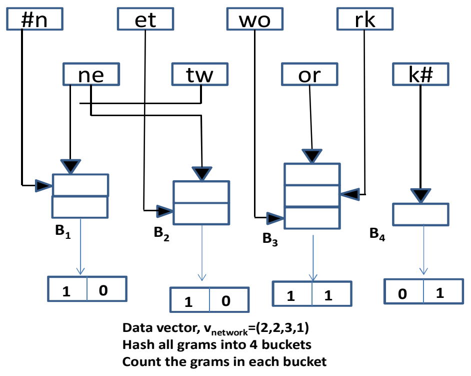
#n
et
wo
rk
ne
tw
or
k#
B1
B2
B3
B4
1 0
1 0
1 1
0 1
Data vector, v_network=(2,2,3,1)
Hash all grams into 4 buckets
Count the grams in each bucket

Fig 2: L buckets for n-gram insertion

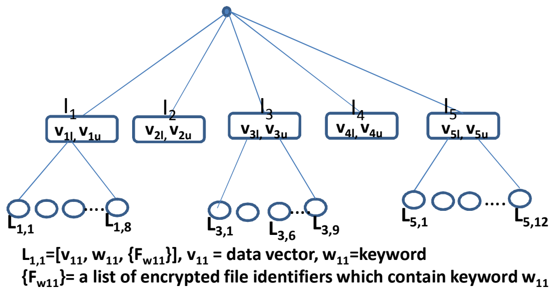
I1
v1l, v1u
I2
v2l, v2u
I3
v3l, v3u
I4
v4l, v4u
I5
v5l, v5u
L1,1
L1,8
L3,1
L3,6
L3,9
L5,1
L5,12
L1,1=[v11, w11, {Fw11}], v11 = data vector, w11=keyword
{Fw11}= a list of encrypted file identifiers which contain keyword w11

Fig 3: BedTree Index Tree Construction

In [12], the authors can store the exact keyword in the data structure of an index node within the index tree which facilitates the exact comparison of the query and stored keywords. However, in our application scenario, we cannot store keywords since we need to preserve keyword privacy. Instead, we store hash values of that string. Typically, we only need to store one hash value unless hash collisions with other words occur. When that happens, that keyword will be represented by as many hash values as needed to uniquely identify the string. During the construction of the index tree, if we find that k words map to the same hash value using the first hash function f1, a second (or third) hash function f2 (f3) will be used to hash those k words until they can be uniquely identified. The extra hash values will be stored. Our experience using the two datasets described in Section VI indicates that a 32-bit hash function is sufficient to avoid collisions.

To support fuzzy keyword search, one bloom filter for each edit distance value is constructed for every keyword. This design choice allows us to minimize the collision probability which may cause a fuzzy keyword with a larger edit distance to be found in a bloom filter of another keyword. For example, for the word ***kw*****="network"**, the following words representing all possible keywords having an edit distance of 1 with the word "network":{*network, n*etwork, ne*twork, net*work, netw*ork, netwo*rk, network*k, network*, *etwork, n*twork, ne*work, net*ork, netw*rk, netwo*k, network*} are inserted into a bloom filter, $\boldsymbol{BF_{kw}(1)}$. Then, a data vector, $\boldsymbol{v_{kw}}$ is generated for the word "network". The data vector, $\boldsymbol{v_{kw}}$, its hash value $\boldsymbol{H_{kw}}$ and its bloom filter, $\boldsymbol{BF_{kw}(1)}$ and a list of file identifiers that contain this keyword constitute a new leaf node structure that needs to be inserted into the Bedtree index tree. Typical B+ index tree insertion technique as described in [12] is used to insert this leaf node. For example, in Fig 4, we show an example of an existing index tree. The data vector of a new leaf node is determined to fall within the string interval stored in the third intermediate node denoted as I_3 and finally inserted in the leaf node level as $L_{3,6}$.

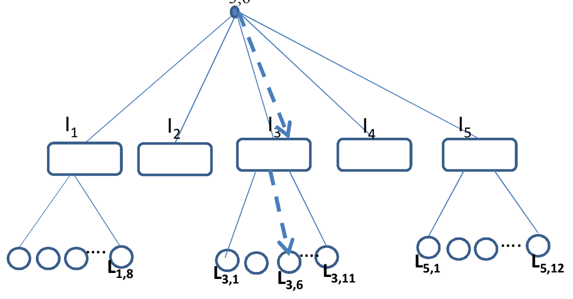

Fig 4: Inserting a new leaf node into the index tree

V. CONSTRUCTIONS OF FUZZY MULTIKEYWORD SEARCH

Here, we describe how we build our system that supports multi-keyword search over encrypted cloud data.

First, the data owner should extract distinct keywords that describe the data items that they wish to publish. For example, if the data items are scientific publications, the keywords can be extracted from the title, the keyword/general term fields, and the abstract of a publication. After extracting distinct keywords set W from the document collection F, the data owner can use domain specific sources to add additional semantic related words to the keyword set e.g. one can use the words for describing the 23 categories from the Microsoft Academic Search domain page [13] as additional keywords for ACM publications. In addition, one can use typical semantic grow bag algorithms e.g. [14] to discover semantically related words to enhance the keywords set. In this work, we merely use (a) co-occurrence probabilities of words present in the core keyword sets to identify relevant double-word keywords e.g. "congestion control", "information retrieval", etc to be added to the keyword set, and (b) the multi-keywords specified in the general and keyword fields of ACM publications are also added.

For each keyword $w \in W$, we construct its associated fuzzy keyword sets $\{\Gamma_w(i)\}$ with edit distance $i=1, \ldots d$. For each constructed fuzzy keyword set $\Gamma_w(i)$ with edit distance i, a bloom filter is constructed $\mathrm{B}_w(i)$ based on the words in the associated fuzzy keyword set. The hash value of the keyword, a list of encrypted file identifiers that contain this keyword, and a list of bloom filters with appropriate edit distances are then inserted into the index tree. A separate bloom filter is used for each edit distance value because we want to minimize the probability of finding the hash value of a word in the fuzzy keyword set of an irrelevant keyword y with edit distance larger than 1 in a bloom filter of a keyword w that contains the hash values of all words in its fuzzy keyword set that has an edit distance of k $(1<k<=d)$.

A. *Basic Search Scheme*

Setup: after extracting distinct keywords set W from the document collection F, the data owner first generates a Bloom filter $\boldsymbol{b_{wi,d}}$ that include the fuzzy keyword set $\boldsymbol{S_{wi,d}}$ for each keyword $\boldsymbol{w_i}$ with edit distance $\boldsymbol{d}$. Then, he generates a M-bit hashvalue, $\boldsymbol{h_{wi}}$ that respresents the keyword, produces the data vector based on the gram counting order [12] and inserts a leaf node containing the following information: $\boldsymbol{[h_{wi}, b_{wi,1}, b_{wi,2}, b_{wi,d}, \{FID_{wi}\}]}$ into the bed-tree structure described in Section III.D. The data owner also encrypts $\boldsymbol{FID_{wi}}$ as $\boldsymbol{Enc(sk, FID_{wi} \| wi)}$. All encrypted file identifiers that contain a keyword will be included in a list, and the address pointer to this list is stored as part of the entry in the leaf node of that constructed index tree. Both the index tree, and the encrypted file identifiers, encrypted documents are then outsourced to the cloud server.

Fig 5 shows how a user submits a fuzzy multikeyword search query. A search request consists of the following tuple $(v_{skey}, h_{skey}, \{BF_{skey,1}, BF_{skey,2}, ..BF_{skey,d}\}, d)$ where *skey* is the keyword that the user is interested in searching, v_{skey} is the data vector based on gram counting order of that keyword, *skey*, d is the edit distance that is tolerable to the user and $BF_{skey,k}$ is a set of hash values that represent the fuzzy keyword set with edit distance k from the word skey. Upon receiving the search request, the server uses the data vector of that keyword to search for the presence of such an entry in the index tree. If the query data-vector and the stored data-vector has edit distance that is within the edit distance bound, then the hashvalues are compared to see if there is any match. If there is a match, the list of stored file identifiers are returned. If there is no exact match, then the server determines if any hash value of the words in the fuzzykeyword set in the query can be found in the bloom filter of any stored leaf node. If there is a match, then the server will retrieve the list of encrypted file identifiers, and include it as part of the response to be sent to the user.

Due to the property of Bloomfilter, there exists non-zero probability of falsely recognizing unrelated words . The probability of a false positive is $f = (1-(1-1/m)^{kn})^k \approx (1-e^{-kn/m})^k$ where m is the bit length of the bloom filter, k is the overal number of keywords in the fuzzy keyword set and n is the # of hash functions utilized in this bloom filter.To minimize the possibility of returning keywords that are not relevant, one should use two or more hash functions to build bloom filters of different edit distance values for each keyword. In addition, instead of using only one data vector, one can use two functions to generate two data vectors to further reduce the collision probability. Our experience using the two datasets described in Section VI.A shows that using two data vectors and two hash functions for the bloom filter is sufficient to reduce this probability to a very small number. The search cost associated with our solution is $O(|W|)$ where W is the number of keywords.

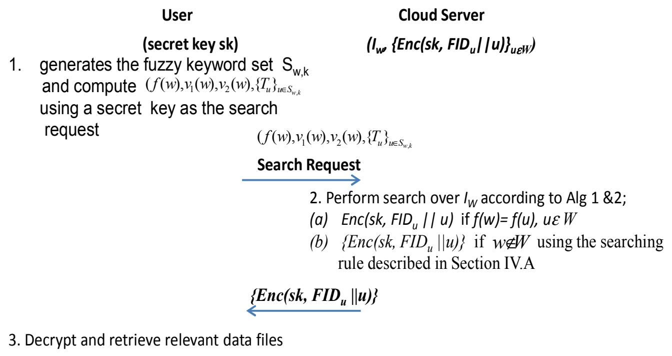
User
(secret key sk)
Cloud Server
$(I_w, \{Enc(sk, FID_u || u)\}_{u \varepsilon W})$
1. generates the fuzzy keyword set $S_{w,k}$ and compute $(f(w), v_1(w), v_2(w), \{T_u\}_{u \in S_{w,k}}$ using a secret key as the search request
$(f(w), v_1(w), v_2(w), \{T_u\}_{u \in S_{w,k}}$
Search Request
2. Perform search over I_W according to Alg 1 &2;
(a) $Enc(sk, FID_u || u)$ if $f(w)= f(u)$, $u \varepsilon W$
(b) $\{Enc(sk, FID_u ||u)\}$ if $w \notin W$ using the searching rule described in Section IV.A
$\{Enc(sk, FID_u ||u)\}$
3. Decrypt and retrieve relevant data files

Fig 5: Protocol for Fuzzy Multikeyword Search

Discussion of the Correctness of the Basic Search Scheme
Theorem 1 in [9] shows that the wildcard-based scheme satisfies both completeness and soundness. Specifically, upon receiving the request w, all of the keywords $\{w_i\}$ will be returned if and only if $ed(w, w_i) <= k$. Similar arguments will work for our basic scheme assuming that the collision probability in the various bloom filters is very small (e.g. 10^{-3}). Below, we give a rough sketch of why we believe our basic search scheme does address the threat model described in Section III.B.
Lemma 1: the intersection of the fuzzy sets $S_{u,k}$ and $S_{w,k}$ for u and w is not empty if and only if $ed(w, u)<=k$
Proof: First, we show that $S_{u,k} \cap S_{w,k}$ is not empty when $ed(w,u)<=k$. To prove this, it is enough to find an element in $S_{u,k} \cap S_{w,k}$. Let $u=a_1, a_2, ..a_s$, $w= b_1, b_2, ..b_t$ where all these a_i and b_j are single characters. After ed(u,w) edit operations, u can be changed to w according to the definition of edit distance. Let $u^*= a_1^*, a_2^*, .. a_m^*$, where $a_j^* = a_j$ or $a_j^* = *$ if any operation is performed at this position. Since the edit distance operation is inverted from w, the same positions containing wildcard at u^* will be performed. Because $ed(u,w)<=k$, u^* is included in both $S_{u,k}$ and $S_{w,k}$.

Next, we prove that $S_{u,k} \cap S_{w,k}$ is empty if $ed(u,w) > k$. The proof is given by reduction Assume that there exists a u^* belonging to $S_{u,k} \cap S_{w,k}$. We will show that $ed(u,w)<=k$ which shows a contradiction. First, from the assumption that $u^* \in S_{u,k} \cap S_{w,k}$, the number of wildcard in u^*, n^*, is not greater than k. Next, we prove that $ed(u,w) <=n^*$. We can prove the inequality with the induction method. First, we show that it holds when n=1. There are 9 cases that should be considered. If u^* is derived from the operation of deletion from both u and w, then ed(u,w) <=1 because the other characters are the same except the character at the same position. The other cases can be analyzed in a similar way and hence omitted. Next, assuming that it holds when $n^* = \alpha$, then we will show that it also holds when $n^* = \alpha+1$. If $u^*= a_1^*, a_2^*, .. a_m^* \in S_{u,k} \cap S_{w,k}$, where $a_j^* = a_j$ or $a_j^* = *$, then for a wildcard position t, cancel the underlying operations and revert it to original characters in u and w at this position. Assume two new elements p, q are derived from them respectively. Then, perform one operation at position t of p to make the character the same with that in w, then, p is changed to p^* which has only k wildcards. Thus, $ed(p^*,w) <=\alpha$. Thus, $ed(u,w)<=\alpha+1$. Thus, we get $ed(u,w) <=n^*$. It renders the contradiction that we want because $n^* <=k$.

To ensure that the probability of collision in the bloom filter is low, one can use a sufficiently large bloom filter (say with M-bits) with several hash functions. In our experiments, we find that using a 32-bit bloom filter with two hash functions is sufficient.

B. *Multi-keyword search*

Assume that the published data items are ACM publications and that a user is interested in searching for all publications with the keywords, "information retrieval". In [9], a querying user has to submit two queries, one for the word "information" and one for the word "retrieval", decrypt the two returned lists of file identifiers, then, perform an intersection of the two lists before he can construct a list of encrypted file identifiers to retrieve the relevant encrypted files from the cloud server.

However, in our approach, that user can just submit a trapdoor representing the hashed value of the words "information retrieval" together with the hash values of words in its associated fuzzy keyword set, and the desired edit distance value to the cloud server if he wishes to use the fuzzy keyword search feature. The cloud server will search the index tree according to the procedures described in Alg1 & Alg2 of Fig 6 to retrieve a list of relevant encrypted file identifiers. Additional pseudo codes for constructing storage and search tokens are included in Appendix 1. To provide efficient search, we further employ some optimizations to prune the number of intermediate and leaf nodes that we need to examine during the process of finding relevant leaf nodes to construct a list of relevant file identifiers as part of a query response. For example, in Fig 7, with optimization, one only needs to traverse intermediate nodes I_1, I_3, I_5, and examine some leaf nodes, but skip the intermediate nodes I_2 and I_4 and all leaf nodes under these two intermediate nodes.

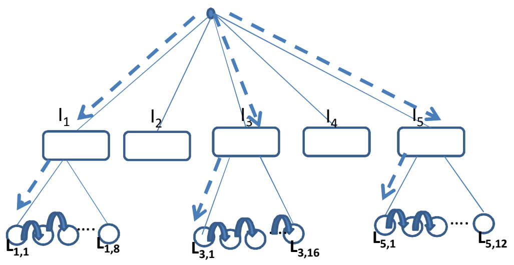

Fig 7: Searching through the index tree

VI. Performance Evaluation

We conduct a thorough experimental evaluation of our proposed techniques on real data sets: the recent 10 years'

IEEE Infocom publications and an extraction of 4888 ACM publications from the ACM digital library. This IEEE Infocom dataset is used by the authors in [9]. The data set consists of about 2600 publications. For each data set, we extract the words in the paper titles to construct the core keyword set in our experiment. In addition, we compute the co-occurrence probabilities of two keywords using the words in the constructed core keyword set to identify extra multi-keywords that can be associated with each title. Using this approach, we can find useful keywords like "power control", "congestion control" etc. The total number of unique keywords for the Infocom dataset is 3278 and their average word length is 7.51. The total number of unique keywords in the ACM dataset is 12386 and the average keyword length is 12.7 (the total length of a multi-keyword is used as the keyword length for computing this average). Our experiment is conducted on a linux machine with an Intel Core 2 processor running at 1.86GHz and 4G DDR2-800 memory. The performance of our scheme is evaluated regarding the time and storage cost of index construction, the search time for 10 queries. We compare our solution with the two approaches described in [9] using the construction time, the storage cost and the searching time as our performance metrics. Since the scheme in [9] does not support multi-keyword directly, we report the total time taken to decrypt the two lists of encrypted file identifiers returned using the two single-word queries.

A. *Performance of Index Tree Construction*

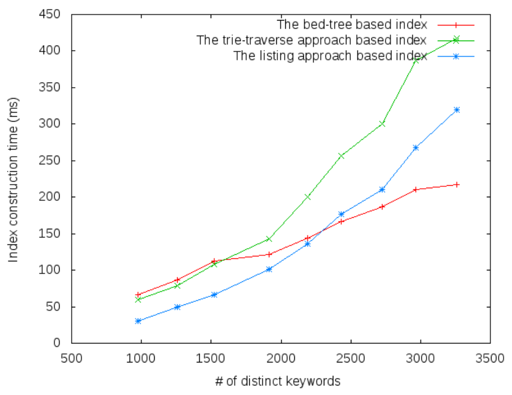

Fig 8: Index Construction Time for our Enhanced BedTree and Wildcard Technique approaches with edit distance d=1

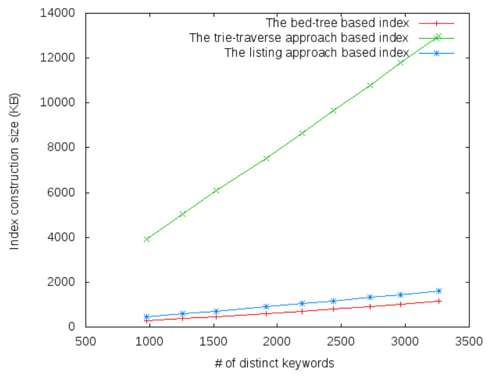

Fig 9: Index Storage Size (Kbytes) for our privacy-aware BedTree and Wildcard Technique approaches with edit distance d=1.

We first investigate how efficient can the index be constructed using our approach. We use SHA-1 as our hash function with an output length of l=160 bits, and take the first 32 bits as the hash value of any keyword. We use different hash functions for data vector and bloom filter generations. Fig 8 shows the index tree construction time (measured in terms of ms) while Fig 9 shows the index storage cost (measured in terms of Kbytes) when we use different numbers of Infocom publication titles (and hence different number of distinct keywords). One can see that our index tree construction time is smaller than the two approaches described in [9] as the number of keywords exceeds 2200. In addition, our storage cost is significantly reduced compared to the symbol-based trie-traversed based approach and slightly better than the listing based approach. Both the construction time and the storage cost increase quite linearly as the number of publication titles increases. Similar plots for the ACM publications dataset are included in Appendix 2. The results for the second dataset equally show that our solution is more efficient.

```
Algorithm 1 VerifyED
Procedure VerifyED (Vector $V_i$, Vector $V_j$)
  Set $m_i = 0$;
  Set $m_j = 0$;
  Set $n$ = the predefined gram length;
  for (k ← 1 to $|V_i|$) do
        if $V_i[k] > V_j[k]$
                $m_i = m_i + (V_i[k] - V_j[k])$;
        end if
          else if $V_j[k] > V_i[k]$
          $m_j = m_j + (V_j[k] - V_i[k])$;
          end if
        end for
  $m_i = m_i/n$;
  $m_j = m_j/n$;
  if $m_i > m_j$
     return $m_i$ as the edit distance;
  end if
  else if $m_j > m_i$
     return $m_i$ as the edit distance;
   end if
End procedure
```

```
      Algorithm 2 Search tree

      Procedure SearchTree (search token $Seartk_{kw} = \{V_{kw}, H_{kw}, \{H(u), u \in S_{kw,\theta}\}$,
      threshold $\theta$, Bed-tree node $N$)
       if N is leaf node then
             for each $Stotk_{kw_i} = \{V_{kw_i}, H_{kw_i}, BF_{kw_i}, p_{kw_i}\}$ in N do
                     if VerifyED ($V_{kw}, V_{kw_i}$) then
                             if $H_{kw_i} = H_{kw}$
                                     include $p_{kw_i}$ in query result;
                             end if
                             else if $H_{kw_i} \neq H_{kw}$
                                     for (i ← 1 to $|HS'_{kw,\theta}|$)
                                       if $HS'_{kw,\theta}[i]$ is found in $BF_{kw_i}$ then
                                       include $p_{kw_i}$ in query result;
                                       Break for;
                                       end if
                                     end for
                             end if
                     end if
              end for
             end if
             else if N is an intermediate node then
               if LowerBound $(V_{kw}, [V_{min}, V_{kw_1}]) \leq \theta$ then
                     Call SearchTree ($Seartk_{kw}, \theta, N \rightarrow 1st\ child\ node$);
               end if
              for (i ← 2 to $|N|$) do
                     if LowerBound $(V_{kw}, [V_{kw_i}, V_{kw_{i+1}}]) \leq \theta$ then
                       Call SearchTree ($Seartk_{kw}, \theta, N \rightarrow ith\ child\ node$);
                     end if
              end for
              if LowerBound $(V_{kw}, [V_{kw_i}, V_{max}]) \leq \theta$ then
                     Call SearchTree ($Seartk_{kw}, \theta, N \rightarrow (|N| + 1)th\ child\ node$);
              end if
         end if
      End procedure
```

Fig 6: Pseudo Code for Searching Index Tree.

B. *Performance of Fuzzy Multi-keyword Search*

Next, we investigate the effectiveness of different searches, and their searching times. Table 2(a) & 2(b) show the search results with one and two data vectors using our approach while Table 3 shows the search results obtained using the wild card technique described in [9]. Table 3 does not include the additional round trip network delay for the multi-keyword queries. Our search time is larger (but still small) with single fuzzy keyword search but the search time for our approach tends to be better for fuzzy multi-keyword queries where many encrypted file identifiers for each word are returned using the single-keyword approach. For example, with the query "congestion control", our search time is 0.6ms while the trie-traverse approach is 1.2 ms. For the query "power control", our search time is 0.3ms while the trie-traverse approach is 0.66ms.

Query Word	Searching time	Returned words
programing	1749 usec	programing
internet	2626 usec	internet
wireless	2826 usec	rulesets, wireless, wireiess
multicast	2108 usec	multicast, mobicast
delay	4354 usec	delav, relay, delays, delay, route, deiay,
differentiated service	546 usec	differentiated service
congestion control	1072 usec	congestion control
power control	3216 usec	power control
expedited forwarding	315 usec	expedited forwarding
proxy caching	1012 usec	proxy caching

(a)Results using one data vector

Query Word	Searching time	Returned words
programing	1186 usec	programing
internet	1468 usec	internet
wireless	1917 usec	wireless, wireiess
multicast	1409 usec	multicast
delay	2764 usec	delav, relay, delays, delay, deiay,
differentiated service	299 usec	differentiated service
congestion control	604 usec	congestion control
power control	2156 usec	power control
expedited forwarding	234 usec	expedited forwarding
proxy caching	656 usec	proxy caching

(b)Results using 2 data vectors
Table 2: Results using Privacy-Aware BedTree Approach

Query Word	Searching time	Returned words
programing	79.11 usec	programing
internet	514.7 usec	internet
wireless	2310.68 usec	wireless
multicast	597.4 usec	multicast
delay	627.66 usec	relay, delays, delay
differentiated service	658.18 usec	differentiated service
congestion control	1193.9 usec	congestion control
power control	1302.3 usec	power control
expedited forwarding	193 usec	expedited forwarding
proxy caching	234.36 usec	proxy caching

Table 3: Fuzzy Keyword Search Using Trie-traversed based Approach.

VII. CONCLUDING REMARKS

In this paper, we have proposed an approach that allows users to conduct fuzzy multi-keyword searches over encrypted data items. Our approach allows easy insertion of newly published data items without having to reconstruct the whole index tree when new information is available. We also use the co-occurrence probabilities to determine additional useful multi-keywords that can be associated with the published encrypted data items. This feature allows us to have faster response times for file retrievals with multi-keyword queries. Our scheme also allows fuzzy multi-keyword search. Our evaluations using two publication datasets show that our proposed solution has better construction time and significantly smaller storage cost as the size of the data files collection increases. In addition, the search time using our approach is reasonable and tends to be better for multi-keyword search where the returned list for individual keyword is large. In the near future, we hope to use more sophisticated techniques e.g. the LDA approach [20] to find semantically related words e.g. "monsters" and "mythical beings" that can be used as additional meta-data descriptions of encrypted data files for fuzzy multi-keyword searches. We also hope to evaluate our solution using other data sets e.g. Enron emails. Last but not least, we hope to develop a prototype, and conduct further evaluations using real cloud services.

ACKNOWLEDGMENT

We would like to thank Cong Wang for sharing his code for the fuzzy keyword search and Z. Zhang [12] for sharing his bed-tree code. We also wish to thank K. Ren for his useful comments on an earlier draft of the paper.

REFERENCES

[1] Google, "Britney spears spelling correction", http://www.google.com/jobs/britney.html, June 2009

[2] M. Bellare, A. Boldyreva, and A. O. Neil,"Determinstic and efficiently searchable encryption", Proceedings of Crypto 2007, LCNS, Vol 4622, 2007.

[3] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data", Proceedings of IEEE Security and Privacy, 2000.

[4] E. J. Goh, "Secure indexes", Cryptology ePrint Archive, Report 2003/216, 2003.

[5] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search", Proceedings of EUROCRYP,04, 2004

[6] B. Waters, D. Balfanz, G. Durfee, and D. Smetters, "Building an encrypted and searchable audit log", Proceeding of 11th Annual network and Distributed System, 2004

[7] R. Cutmola, J. A. Garay, S. Kamara, R. Ostrovsky, "Searchable symmetric encryption: improved definition and efficient constrcutions", Proceedings of ACM CCS, 2006

[8] B. Boneh, B. Waters, "Conjunctive, subset and range queries on encrypted data", Proceedings of TCC, 2007 pp 535-554

[9] J. Li et al, "Fuzzy Keyword Search over Encrypted Data in Cloud Computing", Proceedings of IEEE Infocom, April, 2010

[10] N. Cao et al, "Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data", to appear IEEE Infocom, 2011.

[11] S. Zerr et al, "Zerber: r-confidential indexing for distributed documents", Proceedings of EDBT, 2008 pp 287-298.

[12] Z. Zhang et al, "B^{ed}-Tree: An All Purpose Index Structure for String Similiarity Search Based on Edit Distance", Proceedings of ACM SIGMOD 2010.

[13] Microsoft Academic Search http://academic.research.mcirsoft.com, 2009.

[14] J. Diederich, W.T. Balke, "The Semantic GrowBag Algorithm: Automatically Deriving Categorization Systems", Proceedings of ECDL, 2007.

[15] C. Li, J. Lu, Y. Lu, "Efficient merging and filtering algorithms for approximate string searches", Proceedings of ICDE, 2008

[16] A. Behm, S. Ji, C. Li, J. Lu, "Space-constrained gram-based indexing for efficient approximate string search", Proceedings of ICDE, 2009

[17] S. Ji, G. Li, C. Li, J. Feng, "Efficient interactive fuzzy keyword search", Proceedings of ACM WWW, 2009.

[18] Y. C. Chang, M. Mitzenmacher, "Privacy preserving keyword searches on remote encypted data", Proceedings of ACNS, 2005.

[19] E. Shi, J. Bethencourt, T. H. H. Chan, D. Song, A. Perrig, "Multidimensional range query over encrypted data", IEEE Symposium on Security and Privacy, 2007.

[20] Y. Liu, A. Niculescu-Mizil, W. Gryc, "Topic-Link LDA: Joint Models of Topic and Author Community", Proceedings of International Conference on Machine Learning, June, 2009.

Appendix 1

Algorithm 3: Storage Token Construction

(p_{kw} : a list of pointers that point to the files which contain kw)

```
Procedure CreateStorageToken (keyword $kw$, pointer list $p_{kw}$)
  Construct gram counting order vector $V_{kw}$ for $kw$;
  Hash $kw$ to $H_{kw}$;
  Set BloomFilter $BF_{kw} = \{\emptyset\}$ ;
  Call CreateWildcardFuzzySet ($kw$, $d$) to construct
     FuzzySet $S'_{kw,d}$ for $kw$;
  for (i ← 1 to $|S'_{kw,d}|$) do
     Hash $S'_{kw,d}[i]$ to $H_{kw,i}$;
     Insert $H_{kw,i}$ into $BF_{kw}$;
  end for
  Set $Stotk_{kw} = \{V_{kw}, H_{kw}, BF_{kw}, p_{kw}\}$
   (storage token for $kw$);
        Insert $Stotk_{kw}$ into index tree.
End procedure
```

Algorithm 4: Search Token Construction

```
Procedure CreateSearchToken (keyword $kw$)
Construct gram counting order vector $V_{kw}$ for $kw$;
  Hash $kw$ to $H_{kw}$;
  Call CreateWildcardFuzzySet ($kw$, $d$) to construct
    FuzzySet $S'_{kw,d}$ for $kw$;
    Set $HS'_{kw,d} = \{\emptyset\}$;
    for (i ← 1 to $|S'_{kw,d}|$) do
     Hash $S'_{kw,d}[i]$ to $H_{kw,i}$;
     Set $HS'_{kw,d} = HS'_{kw,d} \cup \{H_{kw,i}\}$;
    end for
    Set $Seartk_{kw} = \{V_{kw}, H_{kw}, HS'_{kw,d}\}$ (search token for $kw$);
End procedure
```

Algorithm 5 Insert Storage Token into Index Tree

```
Procedure InsertTree (storage token
$Stotk_{kw} = \{V_{kw}, H_{kw}, BF_{kw}, p_{kw}\}$, Bed-tree node $N$)
 if N is leaf node then
  for each $Stotk_{kw_i} = \{V_{kw_i}, H_{kw_i}, BF_{kw_i}, p_{kw_i}\}$ in N do
        if $V_{kw} \geq V_{kw_i}$ then
            if $H_{kw} = H_{kw_i}$ then
                 $p_{kw_i} = p_{kw_i} \cup p_{kw}$;
              Break for;
            end if
          else if $H_{kw} \neq H_{kw_i}$ then
            insert $Stotk_{kw}$ into $N$ after $Stotk_{kw_i}$
              Break for;
            end if
            end if
  end for
else if N is intermediate node then
    for (i ← 1 to $|N|$) do
        if $V_{kw} \leq V_{kw_i}$ then
       Call InsertTree ($Stotk_{kw}$, $N \rightarrow ith\ child\ node$);
      Break for;
        end if
    end for
End procedure
```

Appendix 2

Figures 10 & 11 show the index construction time and storage cost of the various approaches using the ACM publications dataset. The results show that our approach is significant better in terms of both construction time and storage cost. The results also indicate that our solution scales linearly with increasing data items.

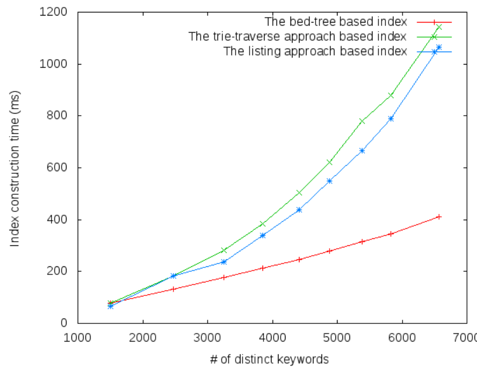

Fig 10: Index Construction Time for our Enhanced BedTree and Wildcard Technique approaches with edit distance d=1 for the ACM publications dataset.

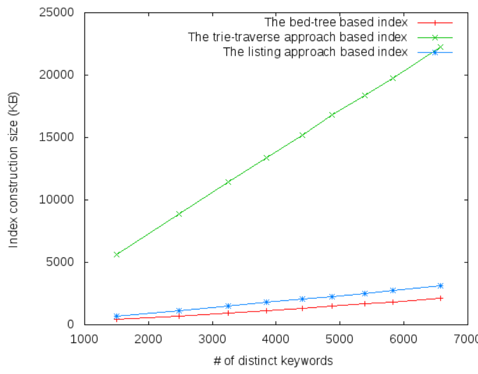

Fig 11: Index Storage Size (Kbytes) for our privacy-aware BedTree and Wildcard Technique approaches with edit distance d=1 for the ACM publications dataset.