Fuzzy Search over Encrypted data using Locality Sensitive Hashing

Madhur Navandar (2020H1030163H)

Abstract-- **Due to the increasing demand for cloud service providers nowadays, data owners stored data in an encrypted format for security purposes. Storing data in the encrypted form leads to an increase in the difficulty of finding particular files consisting of specific keywords, so few searchable encryption techniques are being introduced. A few more techniques like fuzzy search over encrypted data have become a topic of interest for many researchers to improve different search systems. Various methods are introduced to have fuzzy search over encrypted data. Still, few of them lead to leakage of information techniques, either leading to rank leakage of data or leakage of some sensitive data. In this paper, we will focus on searching fuzzy data using locality-sensitive hashing. Locality-sensitive hashing is traditionally used in similarity search, and if the data set size increases, it creates an LSH forest for indexing the keywords. We are using a similar approach in case of our similarity search over encrypted data. We have also discussed our experiment results and the efficiency of this method, such as query search time and index creation cost.**

Keywords— fuzzy search, locality sensitive hashing

I. Introduction

Due to the increase in the volume of data, local organizations outsourced data on the cloud to decrease the cost of managing data on the "on-premise" level. The local organization generally prefers various popular cloud service providers like Amazon, IBM, Google, Azure due to distinctive characteristics such as fault tolerance and easy access anywhere anytime. Though many cloud service providers mentioned data security as their primary concern, organizations generally outsourced data on the cloud in an encrypted format as a safety measure. Still, with data encryption, there are many challenges, like data owners cannot search for the particular query in a file. One of the naïve solution data owners thought of it for performing a search operation, and we can fetch the file on our local system from the cloud, decrypt the file, and finally search for a particular key. But it leads to a lot of work and is very inefficient. Another approach data owners can have is to encrypt the keyword which we want to search. The new encrypted word will match with the word present in the encrypted form on the server. Such a process can be termed as deterministic search keywords, but there is a problem attackers can easily attack deterministic encryption. Deterministic encryption is the attack that often helps the adversary crack the text written in encrypted form. So, to resolve to maintain privacy and maintain a search function searchable encryption scheme was proposed. Searchable encryption can be achieved using two techniques, the first one being symmetric searchable encryption and public key symmetric encryption. In the case of searchable encryption, we have three main components: the first one being data owner who encrypts the data along with some keywords, second one being cloud service provider and third one being data user. Basically, for a construction model using searchable encryption, we have four steps. First, we need to generate a key of size k. Using the key, we can create an index table containing keywords and documents id; Trapdoor, a model which takes query keyword w and generates its ciphertext. Finally, we will use the search method for the search model. But such an approach will only be helpful when the user inputs the exact keyword, and they will get a relevant file containing a keyword. Still, we can extend the approach of searchable encryption, overextending it to finding the relevant document, which is similar to the keyword which user input if the user has made some error while typing a keyword. We need to design an efficient technique for fuzzy search. Apart from similarity, we also have cases in which some approaches are made to split keyword character by character. The searches are done based on the characters present in the keyword. Such an approach sounds straightforward, but it leads to errors when we have keywords whose anagrams are also possible. To improve the drawback, we need to find an order-preserving keyword approach to save the keywords in the index file or enable an ordered preserving search.

In this paper, we proposed a solution of a fuzzy keyword search scheme by using locality-sensitive hashing. My work can be summarized as follows:

1. Instead of finding all the possible combination of the word for fuzzy index search, we are going to find Jaccard similarity, which will find the similarity between two words
2. For indexing of the keyword, we will be using Minhash LSH forest, which internally creates buckets that contain similar words
3. When a user searches for some query, our results will contain a list of FID's which contains an exact keyword or fuzzy keyword.

The rest of the paper is divided as follows. Section II contains various approaches for searchable encryption; Section III contains problem formulations, Section IV contains implementation, and section V results and conclusions.

II. Literature survey

Zhang (2016), along with his colleagues, proposed a secure indexing scheme over encrypted cloud data. In secure indexing, they focus more on the privacy of the index along with the ability to update correctly. They used a wildcard-based technique for indexing. Wildcard-based approach [1] turns out to be an efficient solution for indexing, but it comes with a drawback if the number of files is huge and the number of keywords is also huge, then we need a lot of space to store the index. However, it can be very efficient when the number of files and the keywords are relatively less. Cao's (2019) extended the concept of fuzzy search over encrypted data by introducing "Order preserved uni-gram" [2]. In such schemes, we preserved the order of the character present in a word such that it will help us in cases where there are anagrams of string like "elbow" and "below." For increasing the efficiency of search, Hierarchical Index Tree (HIT) was introduced. To compare the similarity of words in a file with a query, the search must be quicker. For increasing search, "indexing" was introduced. For file indexing purposes, hashing and tree-like data structure was used. Apart from searching accurate words, similar words can also be the result users may want for such cases, fuzzy search is introduced. Cao observed that using local sensitive hashing (LSH), we can map similar words to the same block using hash functions, and to improve the search time, they use Bloom filter data structure which helps to search whether a given keyword is present in a keyword set or not. However, Bloom filters suffer from false-positive results but are advantageous as the error probability is too low and also use hierarchical clustering where similar items are grouped in one cluster. Preserving order will improve the accuracy in cases when we can anagram keywords associated with two different files. Kaur and Bansal (2020) focus on retrieving top k files containing similar multi-keyword using an asymmetric encryption approach [3]. They found out that using an asymmetric algorithm will be more secure than a symmetric approach when it comes to data security, but it requires more CPU cycles. More CPU cycles will be disadvantageous when we need a large number of files to process. In their proposed architecture, they have used the RSA algorithm for encryption and found out that an asymmetric approach will be more secure to prevent unauthorized access. Zheng's (2020) proposed symmetric key predicate encryption scheme, which creates a binary vector of d-dimensional records and B+ tree-based index[4]. For finding similar keywords, they use a Jaccard-based similarity measure. They use the B+ tree to store binary vectors with the respective file identifier and encrypt the binary tree (internal nodes and leaf nodes) before outsourcing the B+ tree index into the cloud. They observed that using such indexing will increase the efficiency of their query processing time. However, the result might change when we use different similarity metrics like cosine similarity. Wang, in 2014, expanded the concept of single keyword fuzzy search to multi-keyword fuzzy search using bloom filters and local sensitive hashing [5]. They suggested using the bloom filter vector as an index and local sensitive hashing to fill bits in the Bloom filter vector. Local sensitive hashing is useful in cases when we need similar hash values for similar words. For each file on the server, they created a bloom filter associated with it. Before inserting value in the bloom filter, they divide the keyword into a bigram vector. If the keyword is misspelled by an edit distance of one or two, we can still have a very closely related bloom filter representation of both words. They found out that using data structures like bloom filter for indexing will help them save a lot of storage space as compared to linear file structure along at the expense of slightly low accuracy due to false positives in bloom filters. Aswani and Shekhar in 2012 introduced the trie-symbol-based technique [6], where a prefix tree is generated to store similar words. The idea using trie data structure, similar words share common nodes, and similar words can be found using depth-first search traversal. In their paper, they have designed an index using the wildcard-based method and trie-based method. They found out that the time cost of designing trie based index is more than a wildcard-based simple listing of keywords. However, the whole index may take more space than a simple listing of wildcard-based search but may be efficient when the edit distance is more than four compared with the simple listing approach. Through various security analyses, they found that fuzzy keywords can show correct results and fulfil the goal of fuzzy search over encrypted data. Jingsha (2018) has proposed an approach that uses a minhash algorithm and their proposal of creating a keyword fingerprint [7] to remove the need to create a separate fuzzy keyword index table. Along with the keyword fingerprint approach, it also uses a tree data structure to store the keywords in the form of a fingerprint. It will also help avoid repeated storing the same prefix already present in the tree. Their approach has introduced a new way of solving search problems in file index using the approximate nearest search method.

In this paper, we will be extending the approach of minhash method by using the existing MinHashLSH and MinHashLSH forest method in case of fuzzy search over encrypted data with the help of file index..

PRELIMINARIES

Locality-sensitive hashing (LSH) was proposed to find the approximate nearest search. LSH is mainly used when brute force searches are not scalable, generally in the application where we need to search on a huge volume of data. LSH generates a similar hash value for the similar words instead of our regular hash function, which might generate a completely different hash value even in the cases wherein two keywords we have an edit distance of just one. So, locality-sensitive hashing will help us in doing a fuzzy search in our system.

A. Shingling

Shingling is a process of dividing the keywords into a set of the length of k. It is observed that similar words generally share more shingles, and so the Jaccard index between two similar words will be higher. But the problem with using only Jaccard similarity is if we need to compare large keywords, which will take a computationally larger time.

B. MinHash

Minhash algorithm solves the problem of computation over larger length strings for Jaccard's similarity. The main idea of using the MinHash algorithm is to calculate the MinHash signature of the strings that will have fixed length and are independent of their size, and Jaccard index can be computed by finding the number of common elements between the two signatures divided by the size of the signature.

C. MinhashLSH

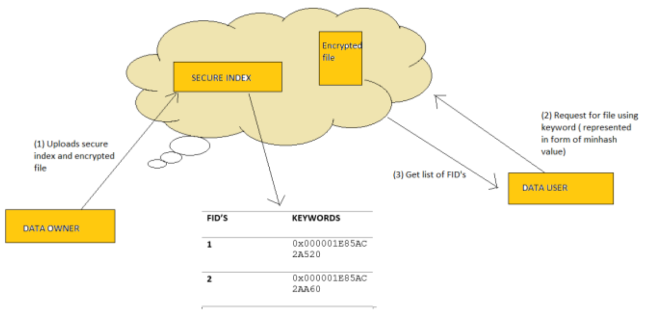
Minhash algorithm can be considered a major improvement in computation time but still if we want to search particular query using MinHash signature, we need to iterate through all the signatures and compare them with the threshold value of similarity. If they match then only, we can say the words are similar. This algorithm will take linear time and will work well for a small number of keywords. So, to reduce time further, Minhash LSH was introduced. The idea of LSH was to hash a particular item several times so that similar words will be more likely to hash to the same bucket compare to dissimilar words. We considered the pair which matches the same bucket as the 'candidate pair' [8]. When we have any query, we will check the candidate pair rather than comparing it with the whole dataset. However, few dissimilar words may hash to the same bucket, so there may be false positives. However, such possibilities are less, and we will have a small fraction of false positives in our results.

D. MinHashLSH Forest

MinHashLSH forest will be useful in cases when we required top k files that have maximum Jaccard similarities. We will use MinHashLSH forest as an index containing FID's and a list of keywords corresponding to them in our system. As keywords will be stored as MinHash values, it can be considered encryption over the index and can also be referred to as 'secure index'.

III. SYSTEM MODEL

In our proposed model, our searchable encryption contains cloud service providers, data owners and data user/authorized data user.


SECURE INDEX
Encrypted file
(1) Uploads secure index and encrypted file
(2) Request for file using keyword (represented in form of minhash value)
(3) Get list of FID's
DATA USER
DATA OWNER

FID'S	KEYWORDS
1	0x000001E85AC2A520
2	0x000001E85AC2AA60

Figure 1: System Model

A. Assumptions

Our proposed scheme will assume that our cloud servers will perform search operations correctly, but there might be a threat to data security or privacy. We are also assuming that the data user is authorized once. Before sending a query to the cloud data, the user must generate minhash value of the query and get a key of the trapdoor. After generating the minhash value, the keyword trapdoor will be sent to the cloud server. Along with keyword trapdoor, data users can also send a value of k so the cloud server will retrieve top-k files from the server and provide it to the data user in encrypted form. Further data user can decrypt the files with key provided by data owner.

B. Proposed scheme

a)Keyword Extraction: For keyword extraction, we are going to use the python package 'yake' [9]. Yet another keyword extractor is After extracting the keyword from files, we will build an index using them

b)BuildIndex : In this step, first, the minhash of the keyword is found and mapped with corresponding FID's. We are using MinHashLSH Forest, the data structure of LSH, for this step.

c)Trapdoor: Data users will create a keyword trapdoor and then send the query to the cloud service provider after converting the query into minhash.

d)Search: After the cloud server receives the query it was searching will be perform using MinHashLSH

IV. IMPLEMENTATION

Following are the steps which are involved in the implementation:

Step 1: First we need to extract keywords from each documents of the file

Step 2: Construct index based on the similarity using MinHashLSH Forest. First, we will split multiple keywords separated by space. Then we will apply tri-gram on each keyword and update its minhash encoding to help in the fuzzy search first. After creating a minhash of a particular word data owner will store it in MinHashLSH Forest in the order of FID's and minhash of particular keywords. MinHashForest will use minhash of the particular keyword and put it in a bucket where other minhash values are mapped with it with a given threshold.

Step 3: Suppose the keyword is 'Software' and human mistakenly wrote 'Softwaree' then to get a similar minhash of two. We can split keyword in trigram and repeatedly update the minhash value of both keywords. Such approach will create a similar hash of both the keywords. Example: Software will be divided into ('Sof','oft','ftw','twa','war','are'} we will create minhash of each will in set and update it to one single minhash. Similarly, for the query word Softwaree it will be divided into {'Sof','oft','ftw','twa','war','are','ree'} So we can see both the set has only one element different so the minhash of both the words will be similar.

V. Results

For implementation purposes we are using actual RFC dataset. For experimentation purpose we are using Windows 10 operating system, the CPU is Intel® Core™ with i5 processor and 8 GB RAM.

First, we need to extract keywords using yake package which extract stop words. Figure 2 show the time it requires to extract keyword with number of files.

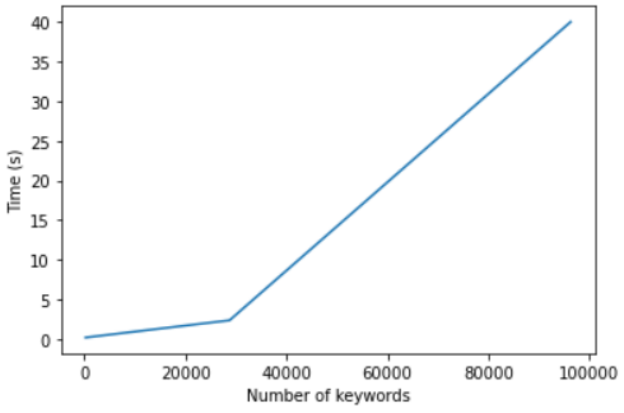


Figure 2: Keyword Extraction

As the average single keyword length for any english word is around four letter so we have decided to use bi-gram and tri-gram based approach in shingling part of our fuzzy system model.

Figure 3 shows the index construction time when we do bi-grams of the keywords and tri-gram of keywords. It can be observed that when we use tri-gram it will take less computation and thus less time.

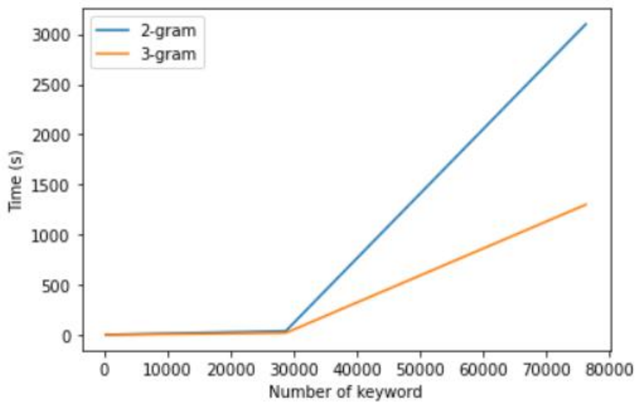


Figure 3 : Index Creation time

Figure 4 shows search time it requires when we use tri-gram and bi-gram. Search time of tri-gram would be slightly better than bi-gram because while searching query needs to first transform into bi-gram or tri-gram respectively as trigram use less time so time requires in case of tri-gram will be slightly less.

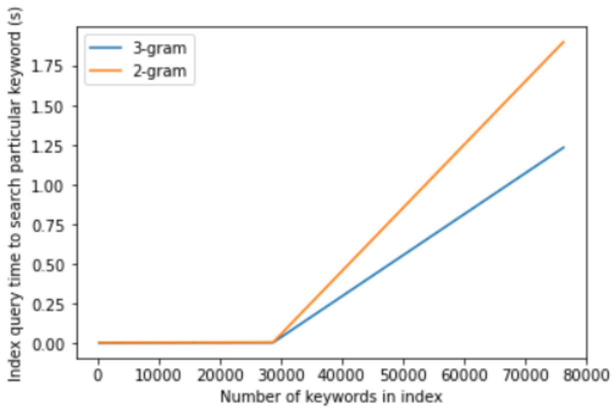


Figure 4: Index Query Time

Performance Evaluation

To evaluate system performance, we are using precision metrics

A. *Precision:* Precision of the model helps us know the what amount of results are actually correct.

Precision: True positive/(True positive + False positive)

B. *Recall* : It helps us find whether our model correctly identify the false positives

Recall: True Positive/(True Positive + False Negative)

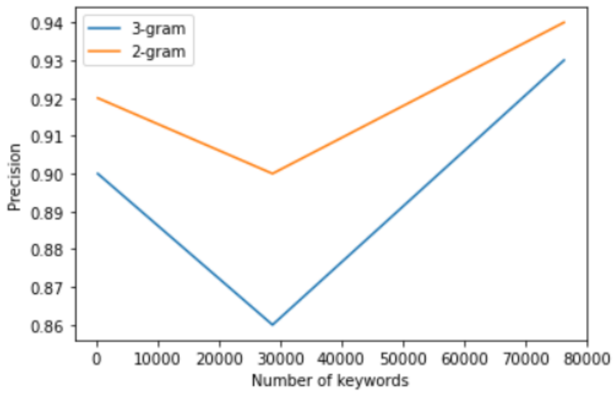


Figure 5: Precision of our model

Figure 5 shows the precision of our LSH based fuzzy search model. As we can clearly see precision of trigram will be slightly lower than bigram because of larger numbers of shingles in bigram.

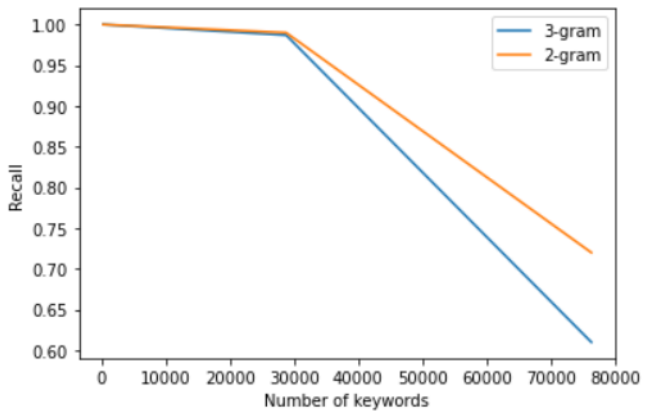


Figure 6: Recall

We can clearly see the recall rate for bi-gram is higher than tri-gram as it will be compute more accurately compare to trigram

VI. Conclusion

The study of searchable encryption and various methods use for data retrieval from the encrypted files stored in cloud server. We have just extended the use of locality sensitive hashing into our problem statement. This section includes conclusion.

The use of storing data in a cloud server acts as a boon for various local organizations. It takes less cost and overcomes the overhead of maintaining data on local on-premise levels. But storing data in a third-party platform may lead to a threat to data security. So data owner uses to store data in an encrypted format, but it leads to searching data as now regular plain-text search won't be possible. So new domain of searchable encryption was introduced. Additionally, to improve search efficiency, further fuzzy search approach over encrypted data becomes a new research topic. This paper has focused mainly on locality-sensitive hashing in our proposed system model initially; minhash of particular keywords was generated, and the minhash value was inserted in the LSH index. Such an index will create a bucket and will place similar items in the same bucket. Apart from that, while presenting the query before the cloud service, keywords from authorized data users are first converted into their minhash values. We have used some python packages for keyword extraction from files and to implement MinhashLSH forest for implementation purposes. After implementation, the system model was evaluated using the actual dataset, and performance was compared. For fuzzy keywords, we need to divide keywords into tri-gram and bi-gram before applying regular minhash. In the results section of this paper, we have compared the difference between using tri-gram based approach and bi-gram based approach. However, both of them have their advantages if the processing of time is not a major concern compare with accuracy we can go for bi-gram based approach; otherwise, use tri-gram.According to performance analysis system model is efficient and can be use in cloud searchable encryption.

References

[1] Baojia Zhang, He Zhang, Boqun Yan, and Yuan Zhang. 2016. A New Secure Index Supporting Efficient Index Updating and Similarity Search on Clouds. In *Proceedings of the 4th ACM International Workshop on Security in Cloud Computing (SCC '16)*. Association for Computing Machinery, New York, NY, USA, 37–43. DOI:https://doi.org/10.1145/2898445.2898451

[2] Cao, Jinkun, Jinhao Zhu, Liwei Lin, Zhengui Xue, Ruhui Ma, and Haibing Guan. "A Novel Fuzzy Search Approach over Encrypted Data with Improved Accuracy and Efficiency." *arXiv preprint arXiv:1904.12111* (2019).

[3] Kaur, Khushpreet and M. Bansal. “Secure Multikeyword Top K Similarity Search Using Asymmetric Encryption Over Encrypted Cloud Data.” (2020).

[4] Campos, R., Mangaravite, V., Pasquali, A., Jatowt, A., Jorge, A., Nunes, C. and Jatowt, A. (2020). YAKE! Keyword Extraction from Single Documents using Multiple Local Features.

[5] Y. Zheng, R. Lu, Y. Guan, J. Shao and H. Zhu, "Achieving Efficient and Privacy-Preserving Exact Set Similarity Search over Encrypted Data," in IEEE Transactions on Dependable and Secure Computing, doi: 10.1109/TDSC.2020.3004442

[6] B. Wang, S. Yu, W. Lou and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," IEEE INFOCOM 2014 - IEEE Conference on Computer Communications, 2014, pp. 2112-2120, doi: 10.1109/INFOCOM.2014.6848153..

[7] Aswani, P. N., & Shekar, K. C. (2012). Fuzzy keyword search over encrypted data using symbol-based Trie-traverse search scheme in cloud computing. *arXiv preprint arXiv:1211.3682*.

[8] He, Jingsha & Wu, Jianan & Zhu, Nafei & Pathan, Muhammad Salman. (2018). MinHash-Based Fuzzy Keyword Search of Encrypted Data across Multiple Cloud Servers. Future Internet. 10. 10.3390/fi10050038.

[9] Campos, R., Mangaravite, V., Pasquali, A., Jatowt, A., Jorge, A., Nunes, C. and Jatowt, A. (2020). YAKE! Keyword Extraction from Single Documents using Multiple Local Features