Montclair State University

Montclair State University Digital Commons

Theses, Dissertations and Culminating Projects

8-2022

Secure Retrieval of Encrypted Similar Documents using Bloom Filters

German Guzman

Follow this and additional works at: https://digitalcommons.montclair.edu/etd

 Part of the Computer Sciences Commons

Abstract

The ability to search for similar documents is a well-known problem on the Web and Information Retrieval field. For example, identifying similar profiles across different government agencies is an important process during intelligence gathering. Nonetheless, when data belongs to multiple parties, internal security policies and government regulations cannot allow the participating parties to freely share their sensitive documents. In our project, we aim to address the following problem: Given a user's query Q and an encrypted database of documents stored on a third-party cloud server, we want to retrieve top-k documents similar to Q without disclosing Q and the contents of the database to the cloud server. We translated documents into bloom filter representations and used the Jaccard Coefficient metric in order to find similarity between each document and Q. We will be conducting empirical tests to validate the reliability, speed, and space efficiency of using Bloom Filters in order to perform operations over Encrypted data. Our proposed solution allows users to keep the contents of documents hidden from unauthorized parties and at the same time facilitating end-users to efficiently retrieve top-k similar documents from the cloud in a secure manner. In our experiments, we found that using a bloom filter provided adequate security amongst the entities involved. The use of bloom filters to measure the Jaccard coefficient accuracy showed good results at a reasonable bit size.

MONTCLAIR STATE UNIVERSITY

Secure Retrieval of Encrypted Similar Documents using Bloom Filters

by

German Guzman

A Master's Thesis Submitted to the Faculty of

Montclair State University

In Partial Fulfillment of the Requirements

For the Degree of

Master of Science

August 2022

College of Science and Mathematics

Department of Computer Science

Thesis Committee:

[redacted]

Dr. Bharath Samanthula

Thesis Sponsor

[redacted]

Dr. Boxiang Dong

Committee Member

[redacted]

Dr. Jiacheng Shang

Committee Member

_____

SECURE RETRIEVAL OF ENCRYPTED SIMILAR DOCUMENTS USING BLOOM FILTERS

A THESIS

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

by

German Guzman

Montclair State University

Montclair, NJ

2022

Copyright © 2022 by German Guzman. All rights reserved.

Acknowledgements

First and foremost, I would like to express my appreciation for all those involved in my master program. I am very grateful to my advisor, Dr. Bharath Samanthula, for helping me with the work behind this master's thesis. I appreciate your countless advice throughout these years. I hope we can continue to work together in the future.

I would like to thank my friends and family for the love and support throughout this academic study. I am fortunate to have a big and loving family that will always be there for me, and I hope I can continue to grow to be there for them like they were for me. I will continue to work hard in everything that I do.

I will cherish everyone I met at Montclair State University. The connections I made in my undergraduate studies are ones I hope I can keep forever. The many friendships and memories made along the way, will always remain with me.

Contents

List of Figures

1. Introduction

1.1 Background and Motivation

The ability to search for similar documents is a well-known problem on the web and Information Retrieval field. The importance of gathering information in a secure way is growing in today's interconnected digital world. When protecting data, one cannot allow the participating parties to freely share their sensitive documents due to privacy issues. These sensitive documents can be critical personal information, also called "personal identifiable information" or "personal health information". This data can include health records, social security numbers, and financial information, such as bank account and credit card numbers. Thus, data privacy is crucial when handling personal data in compliance with data protection laws, regulations, and general privacy best practices.

To achieve privacy, data security must be implemented. However, in software development, speed is a critical factor when trying to reach deadlines or deployment schedules. This can lead to lack of security throughout every step of production process in identifying coding flaws before deploying them into software projects. It is no wonder that according to Forbes, "In 2021, the average number of cyberattacks and data breaches increased by 15.1% from the previous year" [8].

With the increase of data and its need for storage, companies have even outsourced data to the cloud. Cloud computing has become a pivotal tool in the technology field due to its flexibility. Nowadays, the use of cloud computing has grown to where it is easy to use for the average person. Cloud computing has become important for businesses for its easy accessibility, cost savings, and disaster recovery. The ability for users to access their own personal data from anywhere around the world has become the norm. Disaster recovery is also an attribute of cloud

computing. Cloud computing has solidified its role in data backup from individuals to large companies. Having data backed up onto the cloud has given users that extra layer of safety in the case of main systems being damaged.

Finally, the use of cloud computing has become a cheaper way to backup information compared to the cost of using an in-house backup system. With the growth in popularity of cloud computing, the risk of having this data being leaked or compromised can bring major problems.

1.2 Problem Statement

In this paper, we are looking in the scenario of a set of encrypted documents on a cloud. An extra layer of security is provided when encrypting these documents before uploading them onto a cloud service. However, once a document is encrypted, it becomes difficult to identify which document is the one that is needed. A user can choose to perform local encryption, but that would require the user to download all the files from the cloud, decrypt them, until finally being able to read or perform modifications.

In our proposed solution, we use an architecture that involves four entities: the data owner, the client, and two clouds that can be named Cloud 1 and Cloud 2. These four entities collaborate in a system that can search through encrypted documents while keeping all data secure. They are described as the following:

- *The Data Owner:*

  The data owner uploads encrypted documents and encrypted meta-data to Cloud 1. The data owner prepares the documents and encrypts them using AES encryption. On the other hand, the meta-data is encrypted by using pallier encryption which has homomorphic properties.

- *The Client:*

  The client sends queries to the cloud to receive documents of relevance. The client will generate a bloom filter that acts as meta data of the query and is encrypted.

- *Cloud 1 and Cloud 2:*

  These two entities would collaborate in the computation of top-k documents for a given query in a private-oriented way.

The main goal of this proposed solution is to securely compute the top-k documents to avoid disclosing query, Q, and the contents of the database to the cloud server. In the event of a security breach or leak, the contents recovered would be encrypted and the information gathered would be inconsequential. In this paper, we aim to run experiments using bloom filters to test whether it is an accurate and a viable solution for the computation of top-k documents.

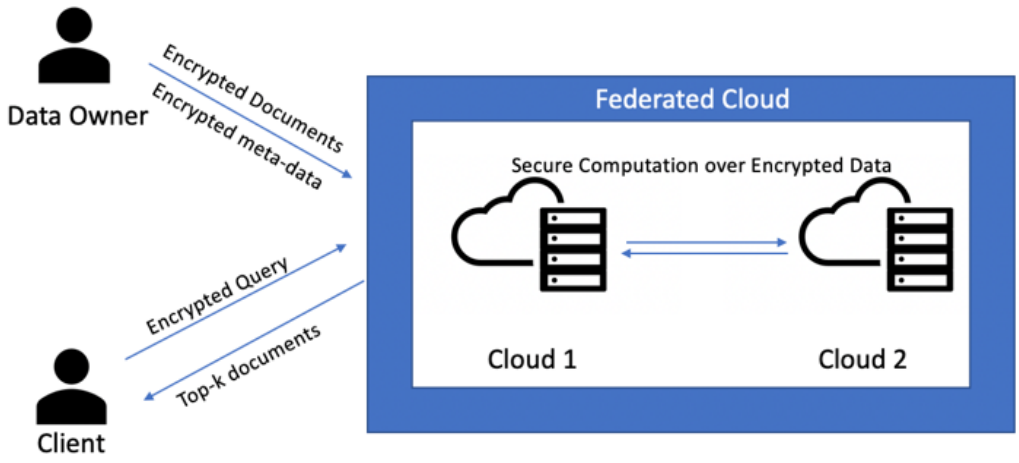

Figure 1: Entities in Proposed Solution

Note. Image of the entities involved in the proposed solution.

2. The Related Work

2.1 Similarity Search

Similarity Search, also known as searchable encryption, is a new developing information security technique that would enable users to search over encrypted data through keywords without having to decrypt the data. One of the first searchable encryption schemes that was proposed was by Song et al. [1], where each word needed to be encrypted with a special construction. Soon after, there were other schemes that used similar index such as Wang et al [4], that used a remote file server and offered solutions, for searchable encryption, under well-defined security requirements. However, this scheme is hindered in the fact that it only supports single keyword search. Like Wang et al., Chauhan [2], also uses LSH with a combination of Shingling and Bloom Filters to reduce search time, respectively to compute Jaccard similarity scores.

Other schemes, such as, Li et al. [5] proposed the use of a fuzzy keyword search scheme over an encrypted cloud data. They combined edit distance with wildcard-based technique to construct fuzzy keyword sets. This scheme managed to address problems such as minor typos and format inconsistence in some manner, however, its performance is dependent on the parameter of edit distance.

Fu et al. [3] proposed a document-based similarity search scheme over encrypted cloud document that found the scheme was quite efficient on real-world datasets. Their experiments had transformed documents into a fingerprint with simhash and hamming distance is used at the similarity metric between documents. Finally, their use of trie-based index was adopted to address the top-k problem and thus, search efficiency showed improvement.

3. Preliminaries

In our proposed solution, there were many methods that we used which included Bloom Filters, Jaccard Coefficient, union and intersection of sets, Pallier Encryption, and Secure Multiplication. To begin, the use of bloom filters played a big role in our proposed approach. This method would allow us to find the top-k similar documents while keeping the contents of each file private. In our performance evaluation, there are two experiments that are conducted to analyze the practicality and accuracy of the use of bloom filters.

3.1 Bloom Filters

A bloom filter is a probabilistic data structure used to optimize memory space and have the property membership checking at O(k) time complexity. Invented by Burton Bloom, in the 1970s, “Bloom filters do not store the items themselves and they use less space than the lower theoretical limit required to store the data correctly, and therefore, they exhibit an error rate” [12]. It is a data structure that can tell whether an element may be in a set or definitely isn’t. There can be a trade-off between performance and accuracy (reporting false positives). A bloom filter will only give false positives and never false negatives. That is to say that it would never report that an element isn’t in our bloom filter if it really is in it.

Implementation

An empty bloom filter is a bit array of m bits where all bits are set to 0. To add an element, it is fed to the hash functions to get k bit positions, and the bits at these positions are set to 1. To check whether an element is in the bloom filter, it is passed to the hash functions and if any k bit positions is reported to have a 0 then it is definitely not in the set.

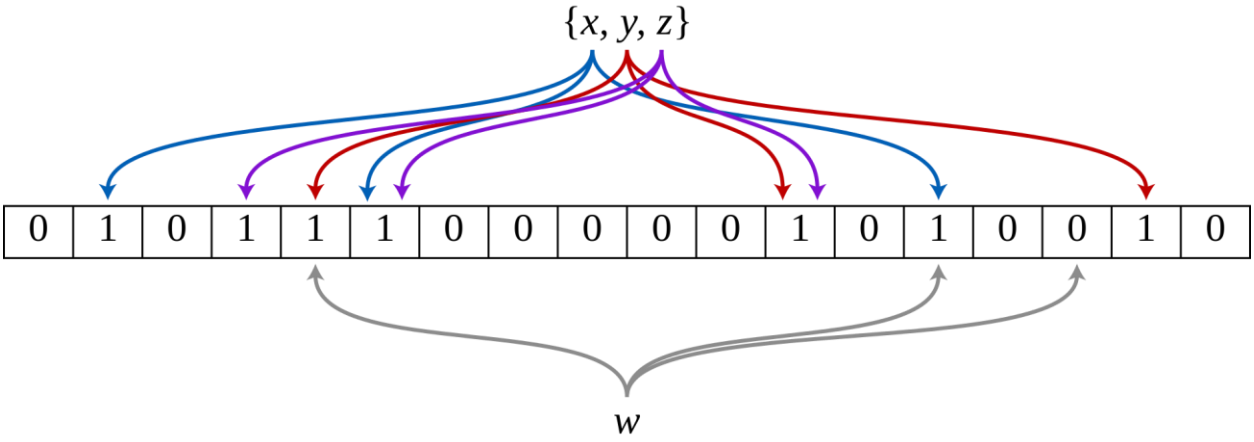

Figure 2: Example of a Bloom filter

Note. An example of a Bloom filter, representing the set {*x*, *y*, *z*}. The colored arrows show the positions in the bit array that each set element is mapped to. The element *w* is not in the set {*x*, *y*, *z*}, because it hashes to one bit-array position containing 0. For this figure, *m* = 18 and k = 3 [6].

The metrics used when constructing a bloom filter are the size of the bloom filter – *m*, the number of hashes – *h*, and the number of top similar documents – *k*. The number of hash functions can also be modified to reduce the probability of collisions or false positives. The number of elements in a bloom filter can be denoted by *n*.

The probability of positive can be controlled by design parameters set upon the bloom filter [7]. By modifying the parameters of number of hash functions and the size of the bit vector, we can reduce the number of collisions. The false positive probability is approximated by the following formula:

$$p = \left(1 - e^{-\frac{kn}{m}}\right)^k$$

The optimal value for the number of hash functions is as follows:

$$k = \frac{m}{n} \ln 2$$

In this formula, we can examine that m/n is the number of bits per element in a bloom filter. Hence, the optimal number of hash functions is linear with the number of bits per element. Lastly, the optimal size for a bloom filter can be as follows:

$$m = \frac{n \log\left(\frac{1}{p}\right)}{(\ln 2)^2}$$

3.2 Jaccard Coefficient

The Jaccard Coefficient (or Jaccard Index) is a statistical metric for gauging the similarity and diversity of sample sets. This similarity index named after researcher, Paul Jaccard, and proposed in 1901, has many applications in set theory. The Jaccard index requires little computational expenses in nature. Luciano da Fontoura Costa writes, “Besides its vast range of applications, most of them related to binary or categorical data, the Jaccard index also motivated some extensions and generalizations, including its adaptation to discrete multisets with positive multiplicites” [11]. In this project, we implement the Jaccard Index score of our sets in plain text and then the encrypted Bloom filters of that sample set to compare the reliability of Bloom Filters. The formula to compute the Jaccard index of two sets can be denoted in Figure 3 [9].

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Figure 3: Jaccard Index Formula

Jaccard Index of sets A and B is the Intersection divided by the Union of the sets.

3.3 Union and Intersection of two sets

When computing the Jaccard Index, the union and intersection between two sets were calculated. Calculating these values for plaintext sets can be easy with Java programming, however, finding the union and intersection of a Bloom Filter sets is different from the formula in Figure 3. Throughout this research we found that there are formulas to find these values in an easier way. To find the union of two bloom filters, we can perform a bitwise OR on every bit. Another way to calculate the union can be denoted as:

$$|B_x \cup B_y| = l - (Number\ of\ zero\ pairs)$$

This formula describes that the union of two bloom filters can be calculated with the length of the bloom filter minus the number of zero pairs (where the value on the two bloom filters is set to 0 on the same index). The length of these bloom filters is computed to be of equal size. One way to calculate the intersection of two bloom filters is to perform a bitwise AND on every bit of

the bloom filters [10]. Another way to calculate this value is to find the union value first, and then use the formula of the intersection of two bloom filters which be denoted in the following:

$$\left| B_x \cap B_y \right| = |B_x| + \left|B_y\right| - \left| B_x \cup B_y \right|$$

One of the key differences between the size of a set and of a bloom filter, is that the size of a bloom filter is the number of 1's contained and not the bit size itself. Hence, in $|B_x|$ and $\left|B_y\right|$, we would need to count number of 1's to find their sizes. The summation of these two sizes is then subtracted by the union value and this would give the intersection of the two bloom filters.

3.4 Pallier Encryption

Pallier Encryption is a well-known public key encryption scheme which also plays an important role in our proposed solution. As being a public encryption scheme, Pallier Encryption also takes advantage of using public and private keys, where the public key is used to encrypt, and the private key is used to decrypt. The Pallier encryption scheme is denoted as:

$$E(m) = g^m \times r^n \, mod \, n^2$$

Where *m* represents the message that is being encrypted, *g* represents the generator, and *r* is the random number. Pallier encryption has two homomorphic properties that we take advantage of, in our project, that differentiates it between other schemes such as RSA. Both properties are the additive properties of homomorphic encryption. The first property is as follows:

$$E(m_1) \times E(m_2) = E(m_1 + m_2)$$

When an encryption scheme is homomorphic, it means that for a fixed key, it is equivalent to perform operations on the plaintexts before encryption, or on the corresponding ciphertexts after encryption [13]. The second property is denoted as $E(m)^c = E(c \cdot m)$. Which is shown by:

$$E(m)^c = \underbrace{E(m) * \ldots * E(m)}_{c}$$

In our proposed solution, we use the Optimized-Pallier. In the Optimized-Pallier, the generator is replaced by (1 + n). This would change the final equation to be denoted as:

$$E(m) = (1 + nm) \cdot R \; mod \; n^2$$

On top of this change, R can also be pre-computed beforehand. In the formula above, R is equal to r^2. The values of r can be created from a table of computed values that are related to r.

3.5 Secure Multiplication

Secure multiplication is a method that involves two parties to get a desired output. These two parties can be denoted as P_1 and P_2. In secure multiplication, P_1 would provide a private input $(E_{pk}(a), E_{pk}(b))$ and P_2 provides a secret key, sk. The idea is for the two parties working together to return $E_{pk}(a \cdot b)$ to P_1 while not revealing any information to the other party [14]. The basis of secure multiplication is derived from the property:

$$a \cdot b = (a + r_a) \cdot (b + r_b) - a \cdot r_b - b \cdot r_b - r_a \cdot r_b$$

P_1 initially randomizes a and b by computing:

$$a' = E_{pk}(a) \cdot E_{pk}(r_a)$$
$$b' = E_{pk}(a) \cdot E_{pk}(r_b)$$

And sends them to P_2, where r_a and r_b are random numbers only known to P_1. After P_2 receives this information, P_2 decrypts and multiplies the value of $h = (a + r_a) \cdot (b + r_b)\ mod\ N$. From here h gets encrypted and sent back to P_1. At this point P_1 removes the random factors from $E_{pk}((a + r_a) \cdot (b + r_b))$ to get the result of $E_{pk}(a \cdot b)$ [14].

4. Overview of proposed solution

This section would include a quick overview of the proposed solution and how we can utilize it to solve the top-k problem. As discussed in the Problem Statement, our solution includes an architecture of four entities which have their own roles. We propose the use of Bloom Filters and Jaccard Coefficient, as well as other methods including, Pallier Encryption and Secure Multiplication.

In this proposed solution, there are two protocols, and each protocol includes three phases. The first two phases are similar in both protocols, but with minor differences in the third phase.

4.1 Phase 1

Phase 1 of the protocols involve data outsourcing. The Data Owner must prepare and send documents to Cloud 1, C_1. Before sending the documents, they need to be encrypted for

security measures. The data owner will use AES encryption to encrypt the documents that are sent to C_1. Each document that is uploaded onto C_1 will be encrypted by the Data Owner with AES encryption key *ek*, where:

$$E_{ek}(D) = (E_{ek}(D_1), E_{ek}(D_2) \dots E_{ek}(D_n))$$

Along with the documents, the Data Owner will upload extra materials, known as meta-data, for each document to C_1. The purpose of the meta-data itself is to make the search for documents easier. To make this step possible, the Data Owner will generate multisets denoted as:

$$M = (M_1, M_2, M_3, \dots M_n)$$

A multiset is an ordered collection of elements where the multiplicity, the number of times that an element appears in a set, of an element matter. These multisets are then translated into multiple sets $S = (S_1, S_2 \dots S_n)$ by finding the unions and intersections of the multisets. A set is different as it doesn't account for the multiplicity of elements.

Next, the sets are translated into bloom filters, $L = (L_1, L_2 \dots L_n)$. The bloom filters themselves act as the meta-data for each encrypted document, $E_{ek}(D)$. The goal is to transfer this data in a secure way to C_1, hence, the meta-data will also be encrypted using the Pallier encryption scheme and its encryption key of *pk*. Thus, the set of all encrypted bloom filters can be denoted as the following:

$$E_{pk}(L) = (E_{pk}(L_1), E_{pk}(L_2) \dots E_{pk}(L_n))$$

Once this is prepared, the Data Owner must send the encrypted document, the encrypted meta-data, and the encrypted size of the document which is part of the meta-data to C_1. There will be a secret key provided by Cloud 2 that will be used for the decryption of this information that is sent.

4.2 Phase 2

The second phase begins with a query being sent by the client, or query sender. The client creates a query, Q which is then encrypted using the public key for Pallier provided by C_2. Based on this query, a bloom filter, L_q, is also created that acts as meta-data of the query. This query bloom filter is also encrypted using Pallier's scheme which results in $E_{pk}(L_q)$ as the query meta-data and the encrypted size of the bloom filter is also sent to C_1.

C_1 will then take the bloom filter of the documents, sent from the Data Owner, $E_{pk}(L_d)$, and the bloom filter of the query, sent from the Client, $E_{pk}(L_q)$, and perform a computation of the Jaccard Coefficient. The computation of this value happens disjointly between C_1 and C_2. A secure multiplication is performed to find the similarity score between $E_{pk}(L_{D_i})$ and $E_{pk}(L_q)$. This similarity is used to find the union size. The secure multiplication would output the number of zero pairs of the two bloom filters which is denoted by T. Once the calculation of zero pairs is confirmed, the union formula is used between the query and document, $E(|D_i \cup Q|) = E(l - (T))$. The calculation of the intersection between these two bloom filters is denoted by $E(|D_i \cap Q|) = E(|D_i| + |Q| - |D_i \cup Q|)$. At the end of this phase, only C_1, knows the encrypted union and intersection sizes of Q and D_i, for $1 \leq i \leq n$.

4.3 Phase 3

To begin, C_1 performs a secure division between the union and intersection is performed to calculate $J(D_i, Q)$. However, due to secure division protocols being inefficient, a different approach is proposed. First, the information that is in C_1 is placed through a random permutation function, π_1, that is provided by C_1 itself. This is done to keep the actual locations of the documents hidden from C_2. The union and intersection values are then sent to C_2 where the decryption process is done using the secret key that is present on C_2. Once decrypted, a division process takes place between the union and intersection values to compute the Jaccard Coefficient. After the Jaccard Coefficient is computed, the locations corresponding to the top-k scores are sent back to C_1 where an inverse permutation function, ${\pi_1}^{-1}$, is done to reveal the actual locations of the top-k documents. Finally, these documents are sent back to the querier.

To avoid the access patterns becoming compromised, another approach is proposed which will be named Protocol 2. As stated before, Protocol 2 follows the same steps for Phase 1 and Phase 2 but has some changes in Phase 3. The differences between Phase 3 begins with performing a random permutation function on the encrypted numerators and denominators and shuffles the data before sending to C_2. The sizes of the numerators and denominators are decrypted using the secret key on C_2 and then the Jaccard Coefficient is calculated. Next, C_2 generates a vector, x, that corresponds to the top-k locations are equal to 1 and those that aren't stay at 0 and is then encrypted, *E(x).* This encrypted vector would then be sent back to C_1 and an inverse permutation function is performed to reveal a vector of actual locations of the top-k documents, y, denoted as:

$$y = {\pi_1}^{-1}(E(x))$$

The vector then goes through an exponentiation involving the encrypted document to give z which is denoted by:

$$z_i = y_i^{E_k(D_i)}, \text{ for } i = 1, \dots, n$$

Finally, the vector goes through a second random permutation function, π_2, and is then sent back to C_2. The vector is then decrypted revealing the non-zero entries in the vector which are the entries of the top-k encrypted documents and are sent back to the client.

5. Performance Evaluation

In this section, we will discuss the two tests that are conducted to analyze the performance of bloom filters and whether it is a viable option. The two tests were divided into two different programs and the results were recorded in a database and put on a graph.

In the two experiments, there were many similarities in the parameters. One of the parameters that really changed the outputs of the program was the bit size of the bloom filter. The accuracy of these bloom filters improved if the bit size was bigger than the possible size of the files themselves. We aimed to push the bloom filters to a small bit size and recorded the accuracy of them.

Before computing the sets, a bit size – *m*, was chosen. The bit size of the two bloom filters needed to be of the same size rather than choosing an optimal size to avoid disclosing any information about the sets themselves. We aim to run our experiments with chosen bloom filter sizes as to keep the memory usage low. Likewise, the number of hash functions was set to 1 for both experiments. This was based solely on the fact that our sets only contained integers and by

adding multiple hash functions, it would make the number of elements in a bloom filter bigger than the sets themselves.

5.1 Experiment #1

For this experiment, the Jaccard Coefficient values between two randomly generated files or sets were analyzed. The files or sets were randomly generated to have a range of 1,000 to 100,000 elements in size. The two values that were compared were of the files' Jaccard index score in plaintext and encrypted version. The goal of this experiment was to measure the reliability of the Jaccard coefficient accuracy while the files were encrypted.

First, the two sets were randomly generated between a number range to simulate real-world file sizes. The sets consisted of positive integers from a number range and contained no duplicate integers. Next, the Jaccard Coefficient Index was calculated in plain text between these two sets. Then, the elements from the sets are hashed and added onto 2 bloom filters. The Jaccard coefficient index is then computed between these two bloom filters. The Jaccard index score of the plaintext sets is compared to the score of the bloom filters and the accuracy is produced. The results can be seen for our first program in Figure 5. As the bloom filter bit size is made smaller, the accuracy decreases. To conclude, the results were very accurate at a size of 100,000 bits and thus, can show reliable results at a reasonable bit size. If a bloom filter is too small, it would have many collisions and report false positives.

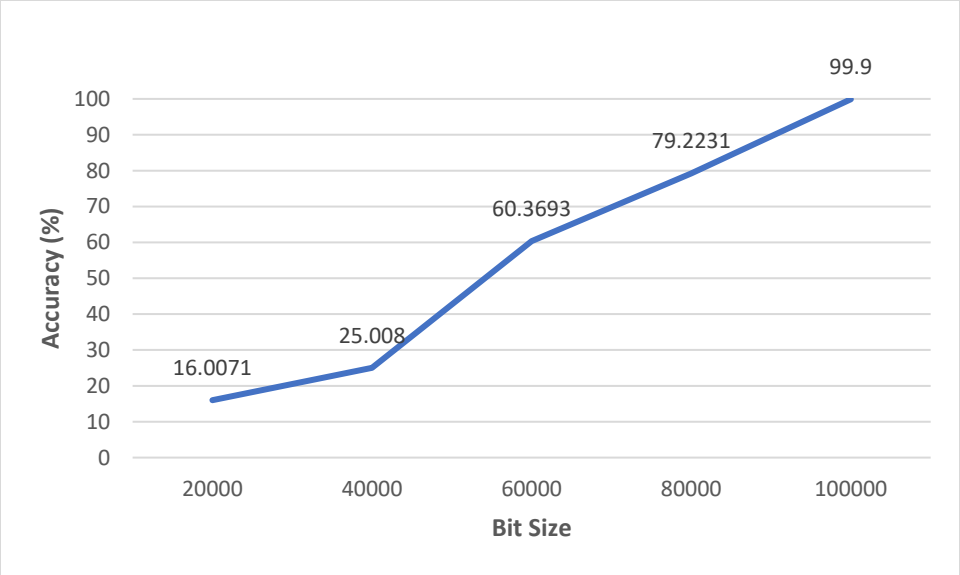

Figure 4: Results for Experiment #1

Note. This graph shows how accurate the estimated Bloom Filter Jaccard index score can be at different bit sizes.

5.2 Experiment #2

In this experiment, we try to measure the accuracy of bloom filters to solve the Top-k problem. Rather than generating two sets, we generate a base set and compare it to 1,000 randomly generated sets stored on a matrix. Next, the sets are hashed and added onto their own respective bloom filters. The Jaccard index score is then computed, between the base set and all the random generated sets, in plain text and hashed bloom filter form. The hashed and unhashed scores, along with their file IDs, are then placed onto two arrays. The top-k similar documents are then computed by sorting the arrays and producing the file IDs with the highest Jaccard index score and an intersection operator is run against them to finally produce the accuracy.

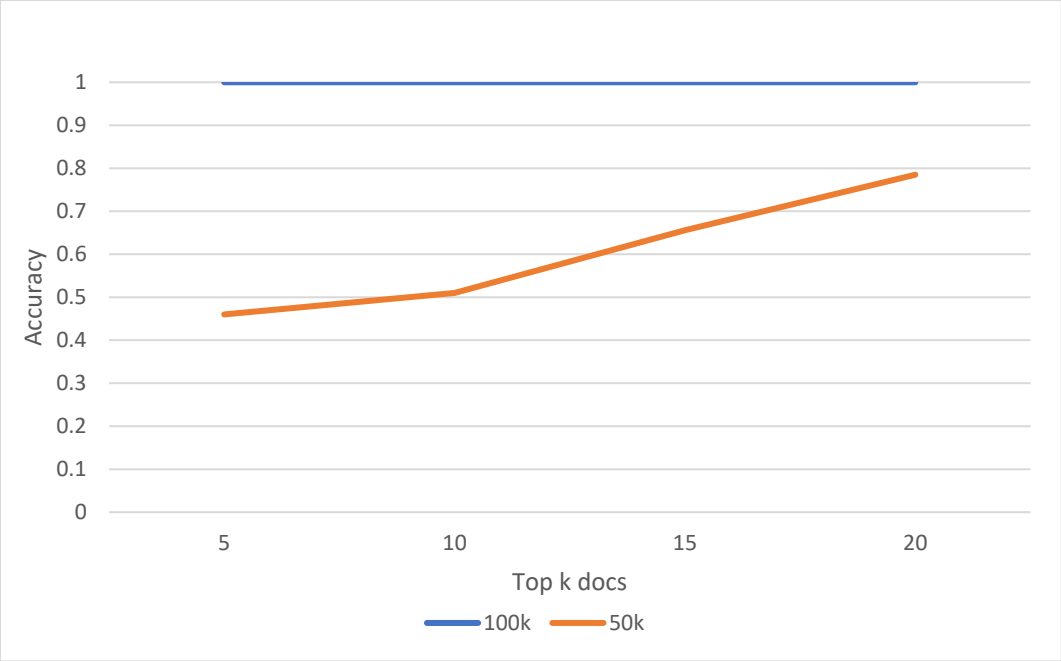

Figure 5: Results for Experiment #2

Note. As the parameter for top-k similar documents is increased, the accuracy is increased. The Blue line shows when the bloom filter bit size is at 100,000 and the orange line shows when the bloom filter bit size is at 50,000.

The results shown in Figure 5, proved that the accuracy produced from Bloom Filters can be dependable when given a reasonable bit size. Increasing the number of top-k documents, also improved the accuracy of the program.

6. Conclusion

6.1 Summary

In this paper, we proposed a secure retrieval of encrypted similar documents using bloom filters. In our solution, we use an architecture that manages to keep user queries and the contents of documents private. Results showed that bloom filters can be a reasonable solution to address

the top-k problem. Experiments on the randomly generated sets proves that bloom filters can be an efficient and realistic solution to find encrypted similar documents.

6.2 Future Work

An outline explaining the future work for the proposed protocol that was presented in this thesis is as follows:

- *Security*: In theory, the proposed protocol is secure and does not disclose its contents amongst the four entities. More experiments need to be done on this protocol to achieve security against leakage or unauthorized users.

- *Experiment variance*: The experiments on files can be expanded to include several types of files such as text documents or spreadsheets. More research can be done to account for different file types.

References

[1] “Practical Techniques for Searches on Encrypted Data.” *IEEE Conference Publication | IEEE Xplore*, 2000, ieeexplore.ieee.org/abstract/document/848445?section=abstract.

[2] “Finding Similar Items Using LSH and Bloom Filter.” *IEEE Conference Publication | IEEE Xplore*, 1 May 2014, ieeexplore.ieee.org/document/7019390.

[3] Fu, Zhang-Jie. “Privacy-Preserving Smart Similarity Search Based on Simhash over Encrypted Data in Cloud Computing.” *Journal of Internet Technology*, 1 May 2015, www.airitilibrary.com/Publication/alDetailedMesh?DocID=16079264-201505-201506030020-201506030020-453-460.

[4] Chang, Yan-Cheng, and Michael Mitzenmacher. “Privacy Preserving Keyword Searches on Remote Encrypted Data.” *SpringerLink*, 2005, link.springer.com/chapter/10.1007/11496137_30?error=cookies_not_supported&code=e32cded3-2fcb-4543-90c2-a0ff37ca57d3.

[5] Li, Jin. “Fuzzy Keyword Search over Encrypted Data in Cloud Computing.” *IEEE Conference Publication | IEEE Xplore*, 1 Mar. 2010, ieeexplore.ieee.org/document/5462196.

[6] Eppstein, David. “David Eppstein Gallery - Wikimedia Commons.” *Bloom Filter Figure*, 13 Mar. 2010, commons.wikimedia.org/wiki/User:David_Eppstein/Gallery#/media/File:Bloom_filter.svg.

[7] Cortesi, Aldo. “3 Rules of Thumb for Bloom Filters.” *Corte.Si*, 25 Aug. 2010, corte.si/posts/code/bloom-filter-rules-of-thumb.

[8] Brooks, Chuck. “Alarming Cyber Statistics For Mid-Year 2022 That You Need To Know.” *Forbes*, 3 June 2022, www.forbes.com/sites/chuckbrooks/2022/06/03/alarming-cyber-statistics-for-mid-year-2022-that-you-need-to-know/?sh=24bec0597864.

[9] DeepAI. “Jaccard Index.” *DeepAI*, 25 June 2020, deepai.org/machine-learning-glossary-and-terms/jaccard-index.

[10] Demofox2. “Estimating Set Membership With a Bloom Filter.” *The Blog at the Bottom of the Sea*, 9 Feb. 2015, blog.demofox.org/2015/02/08/estimating-set-membership-with-a-bloom-filter/#:%7E:text=How%20you%20union%20two%20bloom,Simple%20and%20fast%20to%20calculate.&text=Another%20operation%20you%20can%20do,two%20sets%20have%20in%20common.

[11] Costa, Luciano Da Fontoura. “Further Generalizations of the Jaccard Index.” *Archive ouverte HAL*, 13 May 2022, hal.archives-ouvertes.fr/hal-03384438v4.

[12] Manning Publications. “All About Bloom Filters.” *Manning*, 22 Sept. 2020, freecontent.manning.com/all-about-bloom-filters.

[13] Fontaine, Caroline. “A Survey of Homomorphic Encryption for Nonspecialists - EURASIP Journal on Information Security.” *SpringerOpen*, 11 Dec. 2007, jis-eurasipjournals.springeropen.com/articles/10.1155/2007/13801.

[14] “Secure K-Nearest Neighbor Query over Encrypted Data in Outsourced Environments.” *IEEE Conference Publication | IEEE Xplore*, 1 Mar. 2014, ieeexplore.ieee.org/document/6816690.