ISSN: 2096-3246
Volume 52, Issue 01, Jan, 2020

https://doi.org/10.5281/zenodo.12707009

SECURING ENCRYPTED CLOUD STORAGE DATA BY USING FAST PHRASE SEARCH

K BALAJI SUNIL CHANDRA, JANGILI RAVI KISHORE, DIGALA RAGHAVA RAJU

Assistant Professor [1,2,3]

hod.cse@svitatp.ac.in, jangiliravi.kishore1@gmail.com, raghava.digala@gmail.com

Department of Computer Science and Engineering, Sri Venkateswara Institute of Technology, N.H 44, Hampapuram, Rapthadu, Anantapuramu, Andhra Pradesh 515722

Keywords:
Twitter, Tweets, election, prediction, LDA, Social Influence

ABSTRACT

While cloud computing offers several benefits, it has also raised privacy and security problems, which have piqued the attention of researchers in the field. One of the main issues in this field is the storage and accessibility of sensitive records. Research on methods to decipher encrypted files kept on distant cloud servers has attracted a lot of attention from academics. There have been a lot of proposals for methods to do conjunctive keyword searches, but fewer for methods that are more targeted. In this study, we provide a Bloom filter-based phrase search method that outperforms state-of-the-art methods while maintaining or improving upon storage and transmission costs. To back up the functionality, our method employs a sequence of n-gram filters. The technique is flexible enough to withstand inclusion-relation assaults and displays a trade-off between storage and false positive rate. It also details a design method that takes into account the desired false positive rate of an application.

ISSN: 2096-3246
Volume 52, Issue 01, Jan, 2020
A ES
https://doi.org/10.5281/zenodo.12707009

Introduction

As more and more businesses and people embrace cloud computing, many are becoming aware of the grave privacy and security risks associated with transmitting sensitive data over the Internet. Data breaches, both old and new, have brought attention to the urgent need for safer cloud storage options. Cloud companies often encrypt data and hold the secret keys instead of data owners, even though everyone agrees that encryption is important. That is, consumers have zero privacy protection since the cloud may access whatever data it wants. Another concern in the event of a data breach is the cloud provider's storage of private keys and encrypted data. The need for solutions that keep private keys in the hands of data owners while storing on public and private clouds has prompted academics to actively seek them out. One of the first efforts on keyword searching was proposed by Boneh et al. [1]. Their method enables searchable keywords while concealing their owners' identities via the use of public key encryption. information included within. When it comes to examining encrypted audit logs, Waters et al. [2] looked at the issue. Initial efforts tended to focus on searches with a single keyword. Conjunctive keyword search, which makes use of several terms, has recently seen solutions suggested by researchers [3, 4]. Search result ranking [5, 6, 7] and fuzzy keyword search [8, 9]—a method for searching that may include mistakes—are two more intriguing issues that have been studied. The capacity to look for expressions was also studied not long ago [10], [11], [12], [13]. Some have looked into the safety of the solutions and offered fixes where problems were uncovered [15]. Our word search technique outperforms state-of-the-art solutions in terms of reaction time, and we unveil it in this publication. It is also easy to add or delete documents from the corpus, making the method scalable. We also detail how the strategy was tweaked to protect data from cloud providers that have statistical insight into stored information and to reduce storage costs with no impact on response time. Part 2 lays out the communication structure, and Part 3 provides a variety of contexts, including relevant literature. Even though our method handles phrase searches independently, they are usually only a specialised part of a keyword search system whose main purpose is to provide conjunctive keyword searches. Section 4 therefore details the design methodologies, as well as the fundamental phrase search algorithm and the basic conjunctive keyword search algorithm.Sections 5 and 6 provide the operational evaluation and the outcomes of the experiments. Structure for Communication We'll use the data owner and an unreliable cloud server to illustrate our keyword search methodology. By substituting a proxy server for the data owner and requiring users to authenticate to it, our algorithms may be readily adjusted to suit the needs of a company that wants to set up a cloud server for its workers. Figure 1 depicts a keyword search procedure that is often used. In order to perform hashing and encryption operations, the data owner must first create the necessary keys during setup. The next step is to examine each document in the database for relevant keywords. Hashed keywords and n-grams are connected to bloom filters. After then, the files are sent to the cloud and encrypted using symmetric keys. server. The data owner parses the files during setup and then uploads them to the cloud server with Bloom filters applied to add them to the database. When a data owner wants to delete a file and any associated Bloom filters, they only need to make a request to the cloud server. The data owner computes the searched keywords and transmits them to the cloud in an encrypted form to start the protocol in order to conduct a search.

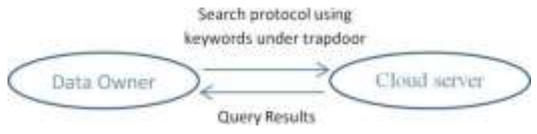

Fig. 1 Communication protocol for searching keywords in encrypted data in order to do a corpus search using the given keywords. As a last step, the cloud provides the data owner with the document IDs they requested. In contrast to previous efforts [1], [2], our architecture uses identity-based encryption and keywords are often

ISSN: 2096-3246
Volume 52, Issue 01, Jan, 2020

https://doi.org/10.5281/zenodo.12707009

meta-data rather than file content. Additionally, we do not utilise a trusted key escrow authority. Recent studies have shown that our scenario is comparable to [10], [16]—in which a company wants to use a cloud storage provider to outsource its computer resources and make search available to its employees—and [6, 17]—in which the goal is to return files that are appropriately rated. Similar approaches, where the client is both the data owner and the user, have been examined in most other recent publications pertaining to search over encrypted data, such as [11]. The encrypted documents may or might not need to be retrieved when the query is resolved; it all depends on the application. Additional privacy concerns can emerge in the event that retrieval is necessary. Opaque storage [18] and private information retrieval methods [19] take these concerns into account. Only the steps leading up to the resolution of the inquiry will be covered in detail. To provide a more accurate comparison with other phrase search solutions, we assume direct retrieval where it makes sense to do so. Safety measures
For the sake of argument, let's pretend that our cloud server is semi-honest; it wants to know more about the data we store on it, but it will use our keyword search protocol exactly as we've defined it and won't alter or falsify any of the data. Concerns about document set privacy and query keyword privacy are two major areas of vulnerability in keyword search security. A secure keyword search protocol, in a nutshell, should make it such that the cloud server can't get any non-neglible information about the documents saved there or the terms used in searches. It should be noted that in our intended application, the users are authorised to search for any document in the data set as they are employees of the organisation that owns the data. In the event that a programme mandates that users not have access to certain files, an access control system like [20] would be necessary to ensure that the matching
outcomes, and only those for which the user has the necessary authorization credentials were returned. Assuming the cloud does not have any previous knowledge of the stored data, our basic strategy in section 4.2 accomplishes these aims. If the cloud service provider knows a lot about the data stored there, including how the keywords are distributed, it may be able to deduce some information about the data's content. In a security model where the cloud provider knows how users search for data, our app needs to make sure that only authorised users can access specific files. To do this, we need an access control system like [20] that checks the results to make sure that only the users with the right credentials can access them. Assuming the cloud does not have any previous knowledge of the stored data, our basic strategy in section 4.2 accomplishes these aims. If the cloud service provider knows a lot about the data stored there, including how the keywords are distributed, it may be able to deduce some information about the data's content. In section 4.4, we detail changes to the fundamental scheme that would mitigate inclusion-relation attacks and statistical assaults, respectively, under the security model in which the cloud provider is aware of the distribution of queries or keywords applied to the stored data.

A ES
https://doi.org/10.5281/zenodo.12707009

ISSN: 2096-3246
Volume 52, Issue 01, Jan, 2020

BACKGROUND

The work of Boneh et al. [1], which proposed a public key encryption-based encrypted keyword search system, was highly referenced in the field. The author took into account a situation when a user wants an email server to confirm messages with certain keywords without disclosing the contents of the emails. In a hypothetical scenario, the system might be implemented such that a user is alerted to an urgent encrypted email and all other emails would be sent to the proper folders. An adaptation using bilinear mapping and identity-based encryption are the components of the suggested system. By looking through encrypted audit logs, only relevant logs are recovered, which is an intriguing use that was proposed by [2]. In this hypothetical situation, an auditor plays the role of a key escrow, granting investigators permission to access audit documents. The plan employs an identity-based encryption extension of Boneh's original technique. After thinking about the situation that Boneh et al. presented, Song et al. [21] came up with a stream cipher-based probabilistic search solution. A lot of new research has concentrated on conjunctive keyword search. In their description of a method that avoided the need of costly pairing operations during encryption and trapdoor generation, Ding et al. [3] built upon Boneh et al.'s strategy by using bilinear mapping to do numerous keyword searches. The search of unstructured text, where the placements of keywords are unknown, was explored by Kerschbaum et al. [4]. In order to protect themselves against chosen keyword attacks, researchers looked into using encrypted indexes for keyword searches in [22]. In [17], Wang et al. examined the ranking of search results. The authors laid up a method that uses order-preserving symmetric encryption in conjunction with the widely-used TF-IDF (Term Frequency x Inverse Document Frequency) rule. Fuzzy keyword search was investigated by Liu et al. [23] as a means of searching for perhaps incorrect terms. The fuzzy dictionaries used by the index-based method include a wide range of misspellings of terms, as well as wildcards. It is only very lately that academics have suggested solutions for phrase searches over encrypted data. In a conjunctive keyword search, as opposed to a phrase search, the query terms must not only be present in the document, but also occur consecutively in the given sequence.

For starters, Zittrower et al. [10] looked into the issue. Two indexes, one for documents and one for keywords' locations, make up his solution. One index shows where in a document you can find a certain keyword, and another shows how to go to that document from the keyword-to-document index. Possible statistical assaults on the indices were discovered by the researchers. The document content may be uncovered by analysing the keyword distribution in the indexes, since some terms are more prevalent in every language. In order to protect ourselves against statistical attacks, we disguised the actual search terms by using truncation of encrypted phrases to create false positives in our query results. Both the index entries and the client-side storage of indicators are used to detect false positives. Due to the significant number of false positives used to offer security, the strategy necessitates a somewhat high computational and communication cost when compared to alternative solutions. The client side also does a lot of the calculation. Phrase search security in a normalization-based method was the primary emphasis of Tang et al. [11]. Additionally, their method makes use of a keyword chain table and an index that maps keywords to documents. The keyword chain table, which is essential for their solution, is used to confirm the presence of keyword pairings. The normalisation of the keyword chain table against all texts in the corpus ensures proven security against statistical assaults. Each item in the table has the same amount of elements since the data used to populate it is randomly generated. Because of this, the data in the table is evenly distributed. Nevertheless, the technique is impractical due to its high storage cost caused by the index tables. Poon et al. [12] offered a different approach that somewhat loosens the security criteria while still achieving much better storage and computational cost. Taking the distribution of keywords in natural languages into account while designing indexes is the key to improvement. Two indices are used, as in [10], [11], to link keywords to documents and their locations, respectively. In order to normalise the data without keeping unnecessary random information, we take into account the almost exponential distribution of keywords and divide the items in the keyword location tables into pairs. Nevertheless, there are certain uses where it could be computationally costly to employ encrypted indexes and execute encryption and decryption on the client side. Further decrease in storage cost was obtained by Poon et al. in [13] using their phrase search system. In order to do conjunctional keyword and phrase searches, the method makes use of the space-efficiency of Bloom filters. A collection of Bloom filters applied to documents

ISSN: 2096-3246
Volume 51, Issue 01, Jan, 2019

https://doi.org/10.5281/zenodo.12707009

and a set of location filters applied to keywords are used, as is the case with previous methods. The former makes it possible to check whether certain documents contain certain keywords. which may be accomplished by just adding the keywords as members, while the latter can be used to identify the locations of keywords by concatenating them with their locations. The storage cost is the lowest among current alternatives, and the approach is theoretically straightforward. Nevertheless, in order to achieve space efficiency, it is necessary to use brute force verification of location during keyword search. The calculation needed increases in direct proportion to the file size since all possible places of the keywords need to be validated. This causes the scheme to have a long execution time.

Bloom filters

Researchers [22], [15], [13] have proposed the use of Bloom filters for conjunctive keyword search to reduce storage cost and provide security in the form of false positives.

Bloom filters are space-efficient probabi Researchers [22], [15], [13] have proposed the use of Bloom filters for conjunctive keyword search to reduce storage cost and provide security in the form of false positives.

Bloom filters are space-efficient probabilistic data struc- tureused to test whether an element is a member of a set. A

listic data struc- tureused to test whether an element is a member of a set. A

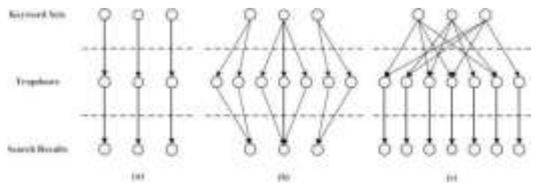

Fig. 2. Relationship between keyword set, trapdoor and searchresult[15

Inclusion-Relation attacks

According to an inclusion-relation (IR) attack published by Cai et al. in [15], it involves two sets of query results, a and b, where an is a subset of b. The idea is that the searched keywords that led to b also contain the ones that led to a. Potentially, some of the keywords may be uncovered by a cloud server that is knowledgeable about the statistical features of the search terms and has access to sets of trapdoors and the search results linked with them.
An adaptation of our method that allows a set of keywords to map to numerous possible inquiries (trapdoors) and includes false positives in search results may be used to guard against such assaults. Furthermore, distinct sets of keywords might result in the same query due to the fact that the same bits in the Bloom filter can be set to 1. Different conjunctive keyword search systems are shown in Figure 2 with their respective mappings of keywords to trapdoors and search results. Our type C design, which was proposed to provide the strongest defence against IR assaults [15], was discussed in section 4.4.
An Approach to Phrase Searching Utilising Bloom Filters
Bloom filters may be used in a keyword search strategy to determine whether a document is connected with a keyword. A location/chain index and a keyword-to-document index are often used in phrase search algorithms [10], [11] to match phrases and map keywords to documents. With a focus on reaction speed, we outline an alternate method that makes use of Bloom filters to bolster this feature. To put it simply, our approach allows for conjunctive keyword and phrase search via the use of several n-gram Bloom filters, abbreviated as Bn.

Search procedure for conjunctive keywords
Each document, Di, is processed for a set of keywords, kwj, in order to provide conjunctive keyword search functionality. At startup, a Bloom filter with a size of m is set to zero. All keywords undergo hashing. generating Hkc (kwj) with a secret key and then passing it through k Bloom filter hash functions to set k bits in the filter. As a consequence, every document will have a 1-gram Bloom filter: Where bi is an element of the set {0, 1}, Di is equal to {b1, b2,...bm}. Securely uploaded to the cloud server using the Bloom filters is

ISSN: 2096-3246
Volume 52, Issue 01, Jan, 2020
https://doi.org/10.5281/zenodo.12707009

the document collection, denoted as D = {D1, D2,..., Dn}. Following this, the Bloom filters are arranged into a matrix with the bit positions in the first row set to Q = q1,{q2,...qx. Then, T = IBF,q1 &IBF,q2...&IBF,qx is calculated, with IBF,qi being the query filter, and the results are sent to the server. on I i, the server is located on the qth row. Documents that are considered to be "matched" are those whose index of bits is set in T. After finding matches, the cloud server may either deliver the document IDs that were matched or, if the application needs it, the encrypted documents. Keep in mind that a conjunctive keyword Bloom filter includes all of a document's keywords, but a query filter only contains a subset of those keywords, so the set Q is much less than m. Hence, compared to individual filter verification, this method may find the matching documents far more quickly with less procedures.
The amount of bits used to represent a Bloom filter index entry is directly proportional to the total number of documents. There are often only a handful of words and bits specified in a query. This causes the matching process to only retrieve a small number of rows. In addition, most computer processors would typically test 32 or 64 bits simultaneously when doing bit-wise AND testing. If a subset of bits in a row returns a test result of 0, then the relevant documents are removed from the candidate pool and the subset of bits does not need to be tested in following rows.

Phrase search protocol

Every document is analysed for collections of keyword pairs and triples in order to provide phrase search capabilities. As an example, the sequence 'Happy Day, Happy Night' would produce the following: 'Happy Day, Day Happy, and Happy Night' as pairs, and 'Happy Day Happy, and Day Happy Night' as triples. For every set of keywords, we calculate a keyed hash, Hkp (kwj kwj+1)|, run it through k hash functions, and then use the output to adjust k bits in the Bloom filter, B2. Triples of keywords are about the first document. The Bloom filter indexes for pairs and triples, IBF 2 and for triples, are generated by transposing the matrices. After matches are found, the cloud server either re-encrypts the documents or returns their matching document IDs, depending on the needs of the application. Two messages are needed to implement our phrase search scheme: the first one is sent to the cloud server with the bit locations of the Bloom filter T query for pairs or triples, and the second one is sent to the data owner with the query results from the phrase search that the cloud performed locally.
The phrase search can only be executed by using k(q 2) hash computations for phrases longer than 2 bits, as well as basic bit-wise AND operations. The computing efficiency of the protocol is high.
It works well with very short phrases and is unaffected by the total number of documents. Our approach also uses less storage than index-based schemes since Bloom filters are space efficient. A scalable solution is provided by the fact that filters are allocated per document. Adding or deleting documents is as easy as adding or removing the related filters. papers that include phrases will always be properly detected, however our approach has the potential to incorrectly identify papers that do not contain phrases. Both the Bloom filter's inherent characteristics and the process used to establish a phrase match contribute to the false positive. Our Bloom filter mechanism kicks in when the user enters n-grams with values of 2 or 3. On the other hand, if n is more than 3, it's feasible that the false positive rate drops to zero, with the exception of those caused by instances when duplicate phrases containing three keywords exist in various sections of a document. Going back to our earlier example of "Happy Day Happy Night," a document would be considered false positive if it included both "Happy Day Happy Day" and "Snowy Day Happy Night" instead of just the phrase. Due to the rarity of such situations in real life, the method is considered legitimate. Aiming for aim accuracy in massive datasets When evaluating a system's efficacy in retrieving and identifying pertinent data, accuracy and recall are often used metrics. The term "recall" refers to the percentage of searched-for documents that were actually found: Think about how many files you intend to receive back for each n-gram query while

ISSN: 2096-3246
Volume 51, Issue 01, Jan, 2019

https://doi.org/10.5281/zenodo.12707009

deciding on a target precision, Precn. B3, the Bloom filter, was also generated by hashing Di. As a consequence Take the case of a query that is anticipated to yield less than five responses; in this case, a There are matrices that organise the bloom filters for pairs and triples. an accuracy rate of 80% would produce On average, there is only one false positive per query, with the first rows including the filters Bx 1. The recall is calculated as TP + TP + FN (3). It can be more appropriate to aim for shorter durations depending on the use case. One definition of precision is the percentage of correctly recognised or retrieved patterns like n - 1, in cases when a larger number of relevant papers for the search: good may provide a challenge ic.

Modified phrase search scheme against IR attacks

(4)The basic scheme can be adapted to provide additional defense

Precision = *TP TP + FP*

They are a way to quantify how well our search algorithms provide relevant results. With our method, we are able to attain a 100% recall rate since there are no false negatives. But since there are more false positives than real positives when searching larger sentences, accuracy drops. As will be shown in section 6.2, the uniqueness of lengthy phrases makes it unlikely that this would severely impact the performance of querying corpuses of normal size. However, it is still not ideal.
protection against inclusion-relation assaults, in which a malicious actor has access to a large database of query Bloom filters and search results linked to recognised queries. Figure 2 shows the modified approach we used to accomplish this. This algorithm was suggested in [15] as a defence against these types of assaults.
The query phrase's beginning and end are stripped of z-terms at random in an IR-secure technique. More words may be deleted to produce false positives since lengthy sentences are unique. Our research led us to this conclusion.

When dealing with very huge datasets, the amount of scientific evidence, which states that z = q/3 | for q↩

6/q and z = q^3 for all values of q

To fine-tune the accuracy rate, bloom filters may be used. By expanding the scheme to include a quadruple Bloom filter (B4) and beyond, at the cost of more storage, it is possible to minimise the amount of false positives. Conversely, reducing the number of filters used may also help with storage reduction. Although it is technically feasible to analyse English texts using only the pairs filter, it is often recommended to include at least one filter for pairs and one for triples due to the high frequency of 2-gram and 3-gram words in the language.
We can calculate the likelihood of a file being matched if we have an application with a target precision, Precn, for n-grams and a corpus containing N documents.

n *Di*

$p_T = 1 \ (\bar{1} \ u_n)^x t - n + 1$, where x^j is the average number of keywords in a file, where u is the likelihood that any two n-grams generated at random will be identical. Next, we aim for a false positive rate of and a true positive rate of around pT N.

I. $\boldsymbol{FP_T} = \frac{p_T N - Prec_n p_T N}{Prec_n}$. (5)

If the likelihood of a file being returned as a false positive is pf, then the likelihood that no more than FPT files were returned as false positives is

$_F\Sigma PT \ .N\ \Sigma$six were successful. In this case, the inclusion link between search words and query results would be severed, leading to an increase in false matches that only comprise sub-phrases. As an example, the query phrase kwj = kw1, kw2, kw3 would be randomly queried as either kwj = kw1, kw2 or kwj = kw2, kw3.

Security

With the EKD (Di) encrypted documents, BDi the conjunctive keyword Bloom filter, and Bn the -giram Bloom filters stored on the cloud server, they are at rest. Reliability and confidentiality of
The symmetric encryption technique guarantees the security of the documents. A secret key is used to hash the words added to the n-gram Bloom filters and the conjunctive keyword Bloom filters in order to prevent the cloud from

ISSN: 2096-3246
Volume 52, Issue 01, Jan, 2020

https://doi.org/10.5281/zenodo.12707009

learning the documents' keywords.
Issues get more complicated when a query is initiated. Using the same secret key for several documents' Bloom filters allows the basic approach to attain great efficiency. As
So, the cloud may check every document in the corpus for the presence of an encrypted keyword or n-gram. The cloud might construct a statistical distribution of encrypted words given enough searches. Could the cloud already know anything about the corpus's statistics, such the fact that the

$p =_{i=1} i \qquad f$

$p^i (1 - p_f)^i$. (6)

language is English or that

it includes papers with a legal standing, it may pick up par-We may answer the following equation for a goal pf if we desire for a probability greater than 90%. With a file containing n-grams and a false positive probability (pf(n,m)) associated with employing an m-gram Bloom filter to verify them, we may calculate the likelihood that the Conjunctive keyword search is supported by the encrypted keyword-to-document index, I. The typical configuration makes use of two sets of n-gram bloom filters, denoted as B2 and B3, to facilitate phrase search. The integration of HK(kwj) is equal to the product of {EK(da, db,..., dn)}. (7) the cloud server receives them. After receiving the encrypted index items, the data owner may decrypt them and use the intersection to find the documents that match: retrieval of the file as a false positive is 1 (one minus one). "If pf(n−,m))fs. • • & DK (I(HK (kw1))) & DK (I(HK (kw2))) If we check that 1 (1 pf(n,m))fs pf, then we may proceed. We want a false positive probability of no more than pf, therefore we add the gramme Bloom filter if the inequality does not hold; otherwise, we meet the aim. The procedure is repeated until the inequality remains true. A combined strategy to counter statistical assaults Conjunctive keyword searches make up the bulk of inquiries in a standard keyword search strategy. Phrase searches are very rare since they are a more specific kind of search.Thus, there would be a lot more statistical data available for individual keywords than for n-grams. Conjunctive keyword matching uses the more secure but costly method of encrypted indexing to protect individual keyword data against statistical assaults. At the expense of client-side encryption/decryption and index re-encryption upon file addition, the method offers information theoretic security for individual keywords [12]. Phrase search with n-gram Bloom filters is still in place. Not only is statistical data scarce because phrase searches are so rare, but there are also many more unique n-grams than keywords [24], leading to a distribution where the individual probability of occurrence is several orders lower than that of keywords [25]. Because there is a huge disparity between the amount of data needed to identify n-grams and the amount of data actually accessible, it is substantially more difficult to launch a statistical assault against n-grams. Our experimental data set displays this quality in Table 1. Important data is affected when files are added to or removed from the corpus. Using unique private keys for each document is a common sense precaution to take against this statistical attack. But there would be a lot of work involved since filters would have to be calculated and checked for each document individually. Rather, as you will see in the sections that follow, we advocate a mixed strategy. it's worth noting that index updates may be postponed to prevent the need to decrypt and re-encrypt the index repeatedly. That is, until the next scheduled index update, the data owner may keep a tiny local index that contains recently added and deleted files. Conjunctive keyword search and phrase search are given different resources in the hybrid technique. With an en- (8) The integral of HK(kwj) is equal to the product of {EK(da, db,..., dn)} because of this.(7) the cloud server receives them. After receiving the encrypted index items, the data owner may decrypt them and use the intersection to find the documents that match: Eighth, using the bitwise AND operation, we get DK (I(HK (kw1)) & DK (I(HK (kw2)) • • • & DK (I(HK (kwq))), (8). The data owner would then start a second chain of communication by passing the document IDs to the cloud server, who would then provide the requested documents if recovery of the encrypted ones is necessary.
for each document, on average. The computation involved in each step requires the cloud server to look up the index entries and calculate hashes of qb bits at certain places. For conjunction keyword searches, the data owner must encrypt q keywords, and for location queries, they must hash one random keyword. Then, in order to create the matching hash signatures, uig encryption and hash calculation of q keywords are necessary. By providing much cheaper cloud storage and eliminating the need to depend on a high volume of false positives to preserve security, the method offers a substantial practical improvement over Zittrower and Tang's

ISSN: 2096-3246
Volume 51, Issue 01, Jan, 2019

https://doi.org/10.5281/zenodo.12707009

approach. Similar to Tang's technique, phrase search is carried out in two stages of communication. The first stage involves finding potential documents that have all of the keywords. To retrieve matching papers, an extra round would be necessary, much as in Zittrower's approach. Noting that more storage reduction is possible, the authors of [13] suggested a storage cost-minimizing technique based on Bloom filters. To allow keyword-to-document search, a conjunctive keyword filter is used for each unique term in a document. To facilitate location inquiries, a phrase search uses a keyword location filter that concatenates terms with their locations. For each document, the technique necessitates two Bloom filters: one to assign keywords to the document and another to calculate the amount of storage required, which is N (bk + bl) bits, where bk is the size of a keyword location. The filters—a conjunctive keyword Bloom filter, a keyword location Bloom filter with a size of bl, and an N-document corpus—are kept on a cloud server. Only the cryptographic keys are kept by the data owner. Excuse me, Concerning the communication cost, the protocol mandates that the keyed hash of the keywords for each filter be sent to the cloud server. Upon receiving the query, the server computes the query filters using qk Bloom filter hashes and returns the results. All of the conjunctive keyword filters in the document set are bitwise ANDed to find the documents that are candidates. Next, in order to find the first word of each phrase in each candidate document, a rk Bloom filter hash computation is required. Then, for each additional word, a rik hash computation is needed to determine if there are any matches. Here, r is the number of keywords in the candidate document and ri is the number of matches for the ith word in the phrase. In most cases, r ri. Hence, phrase searches need about uirk hash calculations, where ui is the number of documents matched during conjunctive keyword searches. The strategy achieved the lowest storage cost among current options because to Bloom filters' space-efficiency, however locating keywords needed a brute-force technique. Despite incremental hash algorithms' ability to speed up verification, the computational cost is still substantial and grows in direct correlation with document size. Due to the significant processing time needed to determine keyword locations, the response time of the scheme suffers, even though the system only needs one round of communication. Our approaches are compared to current methods in Table 3. To make things clearer, we left out a few terms that didn't significantly affect the total cost. It should be noted that the variants, our scheme (storage) and our scheme (speed), are defined in section 6.1, and our hybrid/* scheme corresponds to the hybrid approach described in section 4.6. Our speed scheme has a high storage cost because it takes full advantage of the Bloom filter index to achieve the fastest processing time. However, our storage scheme and variants with different values of t can achieve fast processing time with a storage cost similar to [12] and three times lower than Zittrower's scheme. The method's conjunctive keyword search capabilities is the only way the hybrid approach deviates from the methodology in section 4.2. The phrase search feature is unchanged. Consequently, our base scheme's reaction time, communication cost, and computing cost for phrase search are mimicked. A 1-gram Bloom filter uses less storage space than the system's specific resource for conjunctive keyword search, the index, which is the only real difference.

EXPERIMENTAL RESULTS

Using a corpus of 1500 papers made accessible by Project Gutenberg [26], we assess our method and compare the results to those of current phrase search algorithms. Headers and footers, which include copyright, contact, and source information, were removed during preprocessing of the papers in order to decrease statistical skew. We also do not use stop words. In order to discover the The Natural Language Toolkit [27] was used to assess the statistical features of the corpus as well as the performance of the different strategies. The effectiveness of our method relies on the Bloom filter design. To be more specific, filters must have the same length in order to employ a Bloom filter index. Keep in mind that the following is the formula for the false positive rate given in section 3.1: (p) is equal to (1 − e−k m)k(9). Figure 3 displays the relationship between the amount of bits required for each entry and the number of hash functions, showing false positive rates ranging from 1% to 10%. When considering space requirements, a compact filter is ideal. There would be less need for calculation and communication if the false positive rate was low. The execution time is significantly improved by utilising a minimal number of hash functions, as the computational cost is directly related to their number. Since the false positive rate seldom improves above a certain threshold as the number of hash functions increases, the number of hash functions required to minimise it, denoted as k, is rarely utilised in practice. Sizes of documents using a single hash algorithm, when k= 1. For $k \geq 2$ and $m/n \geq 10$, the

ISSN: 2096-3246
Volume 52, Issue 01, Jan, 2020
https://doi.org/10.5281/zenodo.12707009

false positive rate and the amount of bits per entry remain relatively constant, as seen in figure 4. Deciding on the Size of the Filter If you want to optimise your reaction time, you need to normalise the filter size so you may use the Bloom filter index. As a basic design principle, you should check that the parameters can still fulfil the needs in extreme conditions. The document in the corpus with the highest number of unique keywords or n-grams may be selected as the document with the filter size that meets the application's specified false positive rate. For the vast majority of cases, it would dwarf all other documents in the corpus. The false positive rate for all the other smaller papers would be lower than what is necessary. But there's a hefty storage fee associated with this method. A lot of space goes to waste in databases with widely varying document sizes. If we look at all 15,620 English texts in the Gutenberg corpus, the longest one has 2.8 million words. Half of the texts in the corpus have less than 20,000 words, and only 88 of them have more than 140,000 words. Using t sets of filter sizes, where only one of the document sets would comply to the greatest filter size employed, essentially producing t Bloom filter indexes, allows for a trade-off between response time and storage cost in such cases. In the above example, we can make the biggest 88 documents fit the maximum filter size, the next 7722 documents fit the document with 140,000 words, and the last 7,800 documents fit the document with 20,000 words. Compared to the quickest method, this little tweak would result in a storage demand that is 30 times lower. For this method to work, section 4.2's protocol has to be slightly tweaked so that the k hash functions' output may be transmitted to rather than at the specified bits on the server. By calculating the hash values modulo the filter sizes, the server can easily ascertain the set bit positions. Following each set of filters, the server computes T in the normal way to find matches. Keep in mind that the storage requirements would be lowest if t were set to N, which would mean that the server would have to test each filter individually, but that every document would utilise the precise filter size required to achieve the desired false positive rate. However, when contrasted with current methods, the strategy with t= 1 as "Our scheme (speed)" and t= N time would still provide much better results. In tables 3 and 4, we'll call it "Our scheme (storage)" to emphasise that one of them is optimised for maximum speed, while the other aims for minimum storage cost at the expense of speed. The optimal value of t is application-specific and heavily dependent on the distribution of file sizes. There is no storage benefit to selecting t > 1 if all documents have the same amount of unique keywords or n-grams, as all filter sizes would be equal. After weighing all of the pros and downsides, we settled on an operational p = 5%, k = 2, and m/n = 7.9 for our experimental corpus. Heuristically, we increased the number of bits per entry to m/n = 19, while maintaining the number of hash functions of things that the filters must hold, in order to obtain a more strict false positive rate of 1%. For each conjunctive keyword filter, we would need the amount of bits required to hold the number of different keywords in the biggest document, using the straightforward approach of normalising according to the largest document. The size of each pairs or triples filter is also determined by the number of different pairs or triples. Table 2 displays the values of the parameters for the sample set of Gutenberg documents. Taken together, the 1,530 documents in the Gutenberg corpus allow for a fair comparison to other methods. With a range of 7.9 bits In most cases, the document size is directly proportional to the number of unique words, pairings, or triplets. But this could make it easy for an attacker to spot non-conforming documents if there are outliers. A workaround for this would be to set a minimum ratio of filter size to file size; this would mean that a file with an abnormally low number of unique words would require a greater filter size than is necessary to guard its identity. It is also possible to define a minimum file size so that files with exceptionally large filter sizes are padded to have big file sizes. Extensive expressions Instead of looking for materials on a broad subject, people usually utilise long word searches to find specific things. Finding a certain paper is often the aim. There is a much lower likelihood of occurrence and fewer matches for longer sentences as well. Thus, given a search query with lengthier terms, we would seldom observe more than one false positive, even with a 50% accuracy rate. Our testing showed that inquiries with sentences comprising four or more keywords never resulted in more than one false positive. Additionally, the client-side can easily detect and eliminate the limited number of false positives. So, in theory, lengthier sentences with a lower accuracy rate shouldn't cause any problems. Table 4 provides a summary of the schemes' performance on the sample set of Gutenberg documents using the experimental settings for the parameters listed in table 2. We predicated that, in every instance, the hash values of places and keywords would need 16 bits. The typical length of an English word is five letters. So, 40 is the value of b. For easier comparisons, we preserved dominating words and associated parameters in the formulae since communication

ISSN: 2096-3246
Volume 51, Issue 01, Jan, 2019

https://doi.org/10.5281/zenodo.12707009

and computing costs are query-dependent. When used in real-world situations, ui is greater than u and q, and Enc(x) is less than or equal to Dec(x).

Quantity of unique n-grams for a 150-document sample

Number of words	*n* = 1	*n* = 2	*n* = 3
37626	4833	32023	36884

PERFORMANCE ANALYSIS

Section 4.2 explains that our technique requires two Bloom filters per page for phrase search; one filter stores pairs and the other triples. Keep in mind that phrase searches do not need the use of conjunctive keyword Bloom filters. N bits, where N is the number of documents in the corpus, are needed to store the two sets of filters on the cloud server. Here, b2 is the size of a pairs Bloom filter, b3 is the size of a triples Bloom filter, and N is the absolute value. Cryptography keys are the only things that the data owner has to keep. In our comparison tables 3 and 4, we have included the 1-gram filter, b1, as all current methods have conjunctive keyword search capabilities. Our scheme's communication cost is comparable to that of [13]. Two messages are all that are needed to implement the proposed protocol, with one of those messages carrying the set bit. where the Bloom filter for queries is looking for triples or pairs and the other holds the results that were matched. Presuming the server is down. After receiving the query, the server finds the k(q 2) set bit location by bitwise ANDing the elements of the Bloom filter index, which each have N bits. After that, you may see the matches right away in the outcome. Server and client execution times, message transmission and propagation delays, and other factors all affect the scheme's reaction time. The client's 3(q 2)b keyed hashing is the most costly calculation in the protocol, in contrast to the relatively efficient non-cryptographic hashing and bit-wise AND. The system also needs only one round-trip communication's small propagation latency and has a fast transmission time since the query filter is compactly described. To implement Zittrower's idea [10], an encrypted keyword truncation table is used to assign separate index values to unique keywords that share the same shortened ciphertext. Even though the index table in the cloud stores the same shortened ciphertext, the data owner may use this table to distinguish between entries associated with distinct keywords. This table needs x bits, where x is the number of unique keywords in the corpus, plus b, the average number of bits per term, to have optimum representations. Two index tables are kept on the server in the cloud. One database maps shorter encrypted keywords to documents, while the second table shows where in the papers such keywords are located. In the first table, x(12 + pj log2(N)) bits are needed, while in the second table, xj(12 + gy) bits are needed; here, p is the average number of documents. To implement Zittrower's idea [10], an encrypted keyword truncation table is used to assign separate index values to unique keywords that share the same shortened ciphertext. Even though the index table in the cloud stores the same shortened ciphertext, the data owner may use this table to distinguish between entries associated with distinct keywords. This table needs x bits, where x is the number of unique keywords in the corpus, plus b, the average number of bits per term, to have optimum representations. Two index tables are kept on the server in the cloud. One database maps shorter encrypted keywords to documents, while the second table shows where in the papers such keywords are located. In the first table, x(12 + pj log2(N)) bits are needed, while in the second table, xj(12 + gy) bits are needed; here, p is the average number of documents.Figure 3 shows the relationship between the number of hash functions (k) and bits per entry (m/n) and the false positive rate (p). computing x bits, computing x bits using a Bloom filter hash, computing x bits using a regular hash, looking up x entries in a table, computing the modulus of x numerical values, and bit-wise ANDing x bits. Keep in mind that Zit-trower's approach uses worst-case estimations for its communication and processing costs. Our approaches significantly reduce computing costs for the data owner and the cloud server, as seen in the table. On the cloud, when little operations are required, the benefit is much more pronounced. Additionally,

ISSN: 2096-3246
Volume 52, Issue 01, Jan, 2020

compared to Zittrower's approach, our storage strategy achieves a storage cost that is about five times cheaper. Similar to Tang's method, our schemes only need a single round trip for communication, but with less bits transferred by each side. Notably, the majority of the communication/computation cost is independent of the quantity of keyword matches.

.

CONCLUSION

Our Bloom filter–based phrase search methodology, detailed in this work, outperforms state-of-the-art methods with a single communication round and testing of the filter. Rephrasing phrase search as n-gram verification instead of a location search or sequential chain verification overcomes the significant computational cost stated in [13]. In contrast to the schemes in [10], [12], and [13], our methods do not take the position of a phrase into account; they simply take its presence into account. In contrast to [11], our techniques are parallelizable, have a reasonable storage demand, and do not necessitate sequential verification. Also, unlike previous methods, ours doesn't need a conjunctive keyword search to find potential documents; instead, phrase search may operate autonomously. Method for creating a Bloom filter index as described in Section 4.2

References:

[2] M. Ding, F. Gao, Z. Jin, and H. Zhang, "An efficient public key encryption with conjunctive keyword search scheme basedon pairings," in *IEEE InternationalConference onNetwork Infrastructure and Digital Content*, 2012, pp. 526–530.

[3] F. Kerschbaum, "Secure conjunctive keyword searches for un- structured text," in *International Conference on Network and System Security*, 2011, pp. 285–289.

[4] C. Hu and P. Liu, "Public key encryption with ranked multi- keyword search," in*International Conference on IntelligentNetwork- ing and Collaborative Systems*, 2013, pp. 109–113.

[5] Z. Fu, X. Sun, N. Linge, and L. Zhou, "Achieving effective cloud search services: multi-keyword ranked search over encrypted cloud data supporting synonym query," *IEEE Transactions on Consumer Electronics*, vol. 60, pp. 164–172, 2014.C. L. A. Clarke, G. V. Cormack, and E. A. Tudhope, "Relevance ranking for one to three term queries," *Information Processing and Management: an International Journal*, vol. 36, no. 2, pp. 291–311, Jan. 2000.

[6] H. Tuo and M. Wenping, "An effective fuzzy keyword search scheme in cloudcomputing," in *International Conference on Intel- ligent Networking and CollaborativeSystems*, 2013, pp. 786–789.

[7] M. Zheng and H. Zhou, "An efficient attack on a fuzzy keyword search scheme over encrypted data," in *International Conference on High Performance Computing andCommunications and Embedded and Ubiquitous Computing*, 2013, pp. 1647–1651.

[8] S. Zittrower and C. C. Zou, "Encrypted phrase searching in the cloud," in *IEEE Global Communications Conference*, 2012, pp. 764– 770.

[9] Y. Tang, D. Gu, N. Ding, and H. Lu, "Phrasesearch over encrypted data with symmetric encryption scheme," in *International Confer-ence on Distributed Computing SystemsWorkshops*, 2012, pp. 471–480.