Fast Phrase Search for Encrypted Cloud Storage

[1]VenkataNarayana Gumma, [1]NagaSai Koonisetty, [1]PavanKumar Inagantti, [1]Jagadeesh Jaladi [2] Ramachandran.V,
[1]Student , [2]Professor,[1]Computer Science And Engineering.
[1]Vasireddy Venkatadri Institute Of Technology, Guntur, India

Abstract: —Cloud computing has generated much interest in the research community in recent years for its many advantages, but has also raise security and privacy concerns. The storage and access of confidential documents have been identified as one of the central problems in the area. In particular, many researchers investigated solutions to search over encrypted documents stored on remote cloud servers. While many schemes have been proposed to perform conjunctive keyword search, less attention has been noted on more specialized searching techniques. In this paper, we present a phrase search technique based on Bloom filters that is significantly faster than existing solutions, with similar or better storage and communication cost. Our technique uses a series of n-gram filters to support the functionality. The scheme exhibits a trade-off between storage and false positive rate, and is adaptable to defend against inclusion-relation attacks. A design approach based on an application's target false positive rate is also described.

Index Terms- Conjunctive keyword search, Phrase search, Privacy, Security, Encryption

I. INTRODUCTION

AS organizations and individuals adopt cloud technologies, many have become aware of the serious concerns regarding security and privacy of accessing personal and confidential information over the Internet. In particular, the recent and continuing data breaches highlight the need for more secure cloud storage systems. While it is generally agreed that encryption is necessary, cloud providers often perform the encryption and maintain the private keys in- stead of the data owners. That is, the cloud can read any data it desired, providing no privacy to its users. The storage of private keys and encrypted data by the cloud provider is also problematic in case of data breach. Hence, researchers have actively been exploring solutions for secure storage on private and public clouds where private keys remain in the hands of data owners. Boneh et al. [1] proposed one of the earliest works on keyword searching. Their scheme uses public key encryption to allow keywords to be searchable without revealing data content. Waters et al. [2] investigated the problem for searching over encrypted audit logs. Many of the early works focused on single keyword searches. Recently, re- searchers have proposed solutions on conjunctive keyword search, which involves multiple keywords [3], [4]. Other interesting problems, such as the ranking of search results [5], [6], [7] and searching with keywords that might contain errors [8], [9] termed fuzzy keyword search, have also been considered. The ability to search for phrases was also recently investigated [10], [11], [12], [13]. Some [14] have examined the security of the proposed solutions and, where flaws were found, solutions were proposed [15]. In this paper, we present a phrase search scheme which achieves a much faster response time than existing solutions. The scheme is also scalable, where documents can easily be removed and added to the corpus. We also describe modifications to the scheme to lower storage cost at a small cost in response time and to defend against cloud providers with statistical knowledge on stored data. We begin by presenting the communication framework in section 2 and various backgrounds including related works in section 3. Although phrase searches are processed independently us- ing our technique, they are typically a specialized function in a keyword search scheme, where the primary function is to provide conjunctive keyword searches. Therefore, we de- scribe both the basic conjunctive keyword search algorithm and the basic phrase search algorithm in section 4 along with design techniques in section 4.3. Performance analysis and experimental results are included in section 5 and 6.

II. COMMUNICATION FRAMEWORK

We'll describe our keyword search framework using two parties: The data owner and an untrusted cloud server. Our algorithms can easily be adapted to the scenario of an organization wishing to setup a cloud server for its employees by implementing a proxy server in place of the data owner and having the employees/users authenticate to the proxy server. A standard keyword search protocol is shown in figure 1. During setup, the data owner generates the required encryption keys for hashing and encryption operations. Then, all documents in the database are parsed for keywords. Bloom filters tied to hashed keywords and n- grams are attached. The documents are then symmetrically encrypted and uploaded to the cloud server. To add files to the database, the data owner parses the files as in setup and uploads them with Bloom filters attached to the cloud server. To remove a file from the data, the data owner simply sends the request to the cloud server, who removes the file along with the attached Bloom filters. To perform a search, the data owner computes and sends a trapdoor encryption of the queried keywords to the cloud to initiate a protocol to search for the requested keywords in the corpus. Finally, the cloud responds to the data owner with the identifiers to the requested documents. Our framework differs from some of the earlier works [1], [2], where keywords generally consist of meta-data rather than content of the files and where a trusted key escrow authority is used due to the use of Identity based encryption.

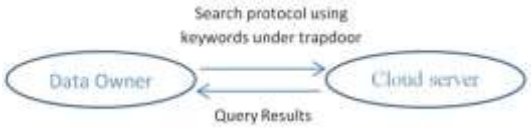

Fig. 1. Communication framework for keyword search over encrypted data

When compared to recent works, our setup is equivalent to that of [10], [16], where an organization wishes to outsource computing resources to a cloud storage provider and enable search for its employees, and similar to [6], [17], where the aim is to return properly ranked files. Most other recent works related to search over encrypted data have considered similar models such as [11], where the client acts as both data owner and user. Note that, depending on the application, the encrypted documents may or may not require retrieval once the query is resolved. Should retrieval be required, further privacy issues may arise. These issues are considered in oblivious storage [18] and private information retrieval schemes [19]. Our discussions will mainly restrict to the protocol leading to the query resolution. Direct retrieval is assumed where appropriate to better compare against existing solutions for phrase search.

2.1 SECURITY

In terms of security, we assume a semi-honest cloud server, which is interested in learning about stored data but will follow our keyword search protocol as described and will not modify or misrepresent any data in order to gain an advantage. Two of the main security issues regarding keyword searches are the privacy of the document sets and the privacy of the queried keywords. Briefly, a secure keyword search protocol should prevent the cloud server from obtaining non-negligible amount of information on the stored documents or the keywords in the query requests. Note that, in our target application, users are employees of the data owner's organization and are authorized to search for any documents in the data set. Should an application requires that users be restricted from accessing certain files, an access control system such as [20] would be required to verify the matched results and returned only those which the user has the required credential to access. Our basic scheme in section 4.2 achieves these goals under the assumption that the cloud has no prior knowledge on the stored data. Should the cloud provider has significant statistical knowledge on the stored data, such as the distribution of the keywords, it may be able to infer partial knowledge on its content. Under the security model where the cloud provider has some knowledge over the distribution of keywords or queries on the stored data, we describe modifications to the basic scheme which would offer protection against statistical attacks in section 4.6 and inclusion-relation attacks in section 4.4.

III. BACKGROUND

Boneh et al.'s work [1] on an encrypted keyword search scheme based on public key encryption was among the most cited in the area. The author considered a scenario where a user wishes to have an email server verify messages associated with certain keywords without revealing the content of the emails. As sample application, the scheme would allow an urgent encrypted email to be flagged to the attention of a user while others sent to appropriate folders. The proposed solution uses identity based encryption and a variant using bilinear mapping. Another interesting application was pro- posed by [2] regarding searching through encrypted audit logs, where only relevant logs are retrieved. The scenario involves an auditor which acts as a key escrow authorizing investigators to search audit records. The scheme uses an extension of Boneh's scheme using identity based encryption. Song et al. [21] also considered the scenario introduced by Boneh et al. and proposed a probabilistic search solution based on stream cipher. Many recent works have focused on conjunctive key- word search. Ding et al. [3] extended Boneh et al.'s scheme using bilinear mapping to perform multiple keyword search and described a solution that did not include expensive pairing operations in the encryption and trapdoor generation phase. Kerschbaum et al. [4] considered the search of unstructured text, where positions of keywords are un- known. The use of encrypted index for keyword search was examined in [22] and a scheme secure against cho- sen keyword attack was proposed. The ranking of search results was looked at by Wang et al. in [17]. The authors described a solution based on the commonly used TF- IDF (Term Frequency x Inverse Document Frequency) rule and the use of order preserving symmetric encryption. Liu et al. [23] considered the search for potentially erroneous keywords termed fuzzy keyword search. The index-based solution makes use of fuzzy dictionaries containing various misspelling of keywords including wildcards. Solutions for searching for phrases over encrypted data were only recently proposed by researchers. The main difference between conjunctive keyword search and phrase search is that the queried keywords must appear contiguously in the specified order in addition to all being present in the document. Zittrower et al. [10] were the first to investigate the problem. His solution uses a keyword-to-document index and a keyword location index. The keyword-to-document index provides the mapping of keywords to the documents which contain them while the location index contains the position of the keywords within each document. The researchers identified potential statistical attacks on the indexes. Since certain words are more common than others in every natural language, the distribution of keywords in the indexes could reveal information on the documents. To defend against statistical attack, truncation of encrypted keywords was used to generate false positives in query results to hide the true search terms. To identify false positives, indicators are included in the index entries and also stored client-side. When compared to other solutions, the scheme requires a fairly high communication and computational cost due to the large amount of false positives used to provide security. Much of the computation is also performed client-side. Tang et al. [11] focused on the security of phrase search in a solution with provable security using normalization. Their technique also uses two index tables: a keyword- to-document index and a keyword chain table. Central to their solution is the keyword chain table used to verify existence of pairs of keywords. In order to achieve provable security against statistical attacks, the keyword chain table is normalized against all documents in the corpus. Random data is used to fill in the table so that the same number of elements is listed under every entry. This results in a uniform distribution of entries in the table. However, the solution has a high storage cost as the index tables require significant storage, which hinders its practicality. Poon et al. [12] proposed an alternative solution which slightly relaxes the security requirements but is able to achieve much improved storage and computational cost. The key to the improvement is the design of indexes con- sidering the

distribution of keywords in natural languages. As in [10], [11], two indexes are used, mapping keywords to documents and keywords to their locations. By recognizing the almost exponential distribution of keywords, the entries in the keyword location tables are split into pairs to achieve normalization without the high cost of storing unused random data. However, the use of encrypted indexes and the need to perform client-side encryption and decryption may still be computationally expensive in certain applications. In [13], Poon et al. proposed a phrase search scheme that achieved further reduction in storage cost. The technique exploits the space-efficiency of Bloom filters to perform con- junctive keyword search and phrase search. Similar to other techniques, a set of keyword to document Bloom filters and a set of keyword location filters are used. The former enables the verification of existence of keywords in individual documents, by simply adding the keywords as members, and the latter allow the identification of keyword locations, by con- catenating keywords to their locations prior to adding them as members. The conceptually simple scheme achieves the lowest storage cost among existing solutions. However, its space-efficiency comes at the cost of requiring a brute force location verification during phrase search. Since all potential locations of the keywords must be verified, the amount of computation required grows proportionally to the file size. As a result, the scheme exhibits a high processing time.

3.1 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 probabilistic data structure used to test whether an element is a member of a set. A Bloom filter contains m bits, where k hash functions, Hi(x), are used to map elements to the m-bits in the filter. The Bloom filter is initially set to all zeros. To add an element, a, to the filter, we compute Hi(a) for i = 1 to k, and set the corresponding positions in the filter to 1. For example, for k = 2 and m = 7, to add 'Bell' to the filter, we compute H1(Bell) = 2 and H2(Bell) = 5.

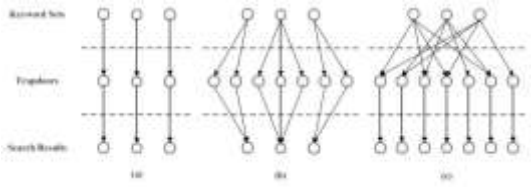

Fig. 2. Relationship between keyword set, trapdoor and search result[15]

Setting the position 2 and 5, the Bloom filter becomes 0,1,0,0,1,0,0. To test for membership of an element, b, in a sample Bloom filter, we compute Hi(b) for i = 1 to k, the element is determined to be a member if all corresponding positions of the sample Bloom filter is set to 1. For example, 'Bell' would be a member of the Bloom filter, 0,1,1,0,0,1,1. While Bloom filters have no false-negatives, it can falsely identify an element as member of a set. Given k hash functions, n items inserted and m bits used in the filter, the probability of false positives is $p = (1\ e^{kn=m})^k$ and minimum false positive rate is achieved when $k = \frac{m}{n} \ln 2$.

3.2 INCLUSION-RELATION ATTACKS

In [15], Cai et al. described an inclusion-relation (IR) attack, where two sets of query results, a and b, with a being a subset of b, would imply the queried keywords that led to b includes keywords that led to a. A cloud server with some knowledge on the statistical properties of the search terms and has access to sets of trapdoors and their associated search results can potentially discover some of the keywords. Our technique can be adapted to defend against such attacks by allowing a set of keywords to map to many possible queries (trapdoors) and the inclusion of false positives in search results. Also, since different sets of keywords can have the same bits in the Bloom filter being set to 1, different keyword sets can lead to the same query. Figure 2 shows different mappings of keywords to trapdoors and search results for various conjunctive keyword search schemes. In section 4.4, we described our type C design, which was suggested to provide the best defense against IR attacks [15].

IV. PHRASE SEARCH SCHEME BASED ON BLOOM FILTERS

In a keyword search scheme, Bloom filters can be used to test whether a keyword is associated with a document. Many existing phrase search schemes [10], [11] use a keyword-to-document index and a location/chain index to map keywords to documents and match phrases. We de- scribe an alternative approach using Bloom filters to support this functionality with an emphasis on response time. Our scheme can be summarized as the use of multiple n-gram Bloom filters, Bn Di, to provide conjunctive keyword search and phrase search.

4.1 CONJUNCTIVE KEYWORD SEARCH PROTOCOL

To provide conjunctive keyword search capability, each document, Di, is parsed for a list of keywords kwj. A Bloom filter of size m is initialized to zeros. Each keyword is hashed using a secret key to produce Hkc (kwj) and passed into k Bloom filter hash functions to set k bits in the Bloom filter. This results in a 1-gram Bloom filter for each document:

BD1i = fb1; b2; :::bmg where bi 2 f0; 1g. The document collection, D = fD1; D2; : : : ; Dng, is encrypted and uploaded

along with the Bloom filters to the cloud server. The Bloom filters are then organized into a matrix with the first row containing the filter BD11 for the first document and the last row containing BD1N . Its transpose is stored as a Bloom filter index IBF where each row corresponds to a bit in the Bloom filters. Note that the ith row in IBF contains information on which document's filter has its ith bit set. This arrangement allows us to quickly identify the documents for a specific query by working only with bits that are set.

To perform a conjunctive keyword search for a set of keywords kw0 = fkw1; kw2:::kwqg, the data owner per-forms the Bloom filter hash computation to determine the set of bit locations, Q = fq1; q2; :::qxg, that would be set in the query filter and sends them to the server. The server then computes T = IBF;q1 &IBF;q2 :::&IBF;qx , where IBF;qi is the qith row in IBF . The index of bits that are set in T are identified as the matched documents. Once the matches are identified, the cloud server can then return the matched document identifiers or the encrypted documents depending on the application requirements. Note that the size of the set Q is much smaller than m since the query filter contains only a few keywords while a conjunctive keyword Bloom filter contains all the keywords in a document. Therefore, this approach can identify the matched documents much faster, performing fewer operations than individual filter verification.

Note that an entry in the Bloom filter index has as many bits as the number of documents. A query generally involves only a few words and very few bits set. These lead to only a few rows being extracted for matching. Furthermore, when performing the bit-wise AND testing, computer processors would generally test 32 or 64 bits at a time. Should a test results in all zeroes for any subset of bits in a row, the corresponding documents are no longer candidates and the subset of bits no longer require testing in subsequent rows.

4.2 PHRASE SEARCH PROTOCOL

To provide phrase search capability, each document is parsed for lists of keyword pairs and triples. For example, 'Happy Day, Happy Night' would yield the pairs, 'Happy Day', 'Day Happy' and 'Happy Night', and the triples, 'Happy Day Happy' and 'Day Happy Night'. A keyed hash for each keyword pair is computed, Hkp(kwj|kwj+1), and passed into k hash functions and the result is used to set k bits in the Bloom filter, B2 Di. Keyword triples are similarly hashed to generate the Bloom filter, B3 Di. The resulting Bloom filters for pairs and triples are organized into matrices with the first rows containing the filters Bx D1 for the first document. The matrices are then transposed to produce the pairs and triples Bloom filter indexes, IBF 2 and

IBF 3, which are stored alongside the encrypted documents on the cloud. To perform a phrase search for the keyword sequence, kw0 = {kw1,kw2...kwq}, the data owner must first per- form the Bloom filter hash computation of the pair, Hkp(kw1|kw2), to determine the set bits in the query filter if the phrase contains two keywords. If the phrase contains more than two keywords, the hashes of triples within the phrase, Hkp(kwj|kwj+1|kwj+2) where j = 1 to q − 2, are evaluated instead. The set bit locations are sent to the server, who then computes T = IBF 2,q1&IBF 2,q2...&IBF 2,qx, where IBF 2,qi is the qth i row in IBF 2 if the phrase contains two keywords, and similarly using IBF 3 for longer phrases. The set bits in T identify the matched documents. That is, for each set bit index, i, in T, the following is true:

$$\{Hkp(kw1|kw2)\} \in B2\ Di \quad (1)$$

for pairs and

$$\{Hkp(kwj|kwj+1|kwj+2)\} \in B3\ Di, \text{ where } j = 1 \text{ to } q-2, \quad (2)$$

for triples.

Once the matches are identified, the cloud server re- turns the matched document identifiers or the encrypted documents depending on the application requirements. Our phrase search scheme requires only 2 messages to be sent: a) The initial message to the cloud server containing the set bit locations of the query Bloom filter T for pairs or triples and b) The response to the data owner containing the query re- sults from the phrase search performed locally by the cloud. Performing the phrase search requires k(q − 2) hash com-putations for phrases of length q > 2 and a simple bit-wise AND operations. The protocol is computationally efficient. Its performance is dependent on the length of the phrase and largely independent of the size of the document set. Due to the space efficiency of Bloom filters, our scheme also requires less storage than index based schemes. Since filters are assigned per document, adding or removing documents consists simply of adding or removing the associated filters, providing a scalable solution. While a document containing a phrase will always be correctly identified as such, our scheme can falsely identify documents as containing a phrase when it doesn't. The source of the false positive is not only the natural property of Bloom filter, but also in how a phrase match is determined. If a user queries n-grams for n = 2 or n = 3, our scheme has no false positives other than ones arising from the use of Bloom filters. For n > 3, however, it is possible that keyword triples within a phrase appear in different parts of a document without the complete phrase being present. Using the previous example of 'Happy Day Happy Night', a false positive would occur if a document does not contain the phrase but instead contains 'Happy Day Happy Day' and 'Snowy Day Happy Night'. The validity of the scheme is based on low occurrence of such scenarios in practical settings.

4.3 DESIGNING FOR TARGET PRECISION IN LARGE CORPUSES

In information retrieval, precision and recall are often used to measure the performance of a system in its ability to retrieve/identify relevant data.

Recall is defined as the fraction of documents relevant to a query that is retrieved/identified in eq(3) and Precision is defined as the fraction of re- trieved/identified documents that are relevant to the query in eq(4).

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4)$$

When applied to our search schemes, they represent a measure of the quality of the matching results. Since our scheme has no false negatives, it achieves 100% recall rate. However, precision tends to decrease when querying longer phrases due to a higher number of false positives relative to true positives. While not ideal, it is unlikely that this would negatively affect the performance of querying corpuses of typical size, due to the uniqueness of long phrases, as will be demonstrated in section 6.2. When working with particularly large corpuses, the number of Bloom filters can be used to tune the precision rate. One may notice that the number of false positives can be reduced by extending the scheme to include a quadruple Bloom filter, B4 Di, and beyond, at the cost of extra storage. On the other hand, one can also reduce storage by using fewer filters. While it's possible to use only the pairs filter, it's generally preferable to use at least one for pairs and one for triples when processing English documents since 2- grams and 3-grams are fairly common in the language. Suppose an application have a target precision, Precn, for n-grams and a corpus with N documents, we can estimate the probability that a file be matched by computing pT = 1−(1−un)x0−n+1, where x0 is the average number of keywords per file and un is the probability of two random n-grams being identical. Then, the number of true positive is approximately pT N and the target number of false positive is

$$FPT = pT\ N - Prec_{npT}\ N\ /Precn. \quad (5)$$

Assume the probability that a file being retrieved as false positive is pf, then the probability that no more than FPT files were found as false positive is

$$p = \sum_{i=1}^{FPT} \binom{n}{i} pi\ f(1 - pf)i. \quad (6)$$

Suppose we wish that the probability p > 90%, the above equation can then be solved for a target pf. Given the probability, $p_{f(n,m)}$, that an n-gram is a false positive when verified using a m-gram Bloom filter and the number of n-grams in a file, fs, the probability that the file is retrieved as a false positive is $1 - (1 - p_{f(n,m)})^{fs}$. If we desire a false positive probability of no greater than pf, then we would verify if $1 - (1 - p_{f(n,m)})^{fs} < p_f$. If the inequality holds, the target is achieved otherwise a m + 1- gram Bloom filter is added. The process repeats until the inequality holds.

Note that when choosing a target precision, Precn, it is helpful to consider the expected number of files returned per n-gram query. For example, if a query is expected to return less than 5 results, a precision of 80% would yield only 1 false positive per query on average. Depending on the application, it may be then preferable to target shorter sequences such as n − 1 where the higher number of false positive could be problematic.

4.4 MODIFIED PHRASE SEARCH SCHEME AGAINST IR ATTACKS

The basic scheme can be adapted to provide additional defense against inclusion-relation attacks where an attacker has access to a significant amount of query Bloom filters and search results associated with known queries. To do so, we modify our algorithm to a type C searching algorithm noted in figure 2, which was proposed in [15] to defend against such attacks. In an IR-secure scheme, z terms are randomly removed from the beginning and the end of the query phrase. Due to the uniqueness of long phrases, more terms can be removed to generate false positives. It was determined, based on our experimental data, that z = bq/3c for q ⩽ 6 and z = q − 3 for q > 6 were effective. For example, a query phrase, kw0 = {kw1,kw2,kw3}, would be queried randomly as kw0 = {kw1,kw2} or kw0 = {kw2,kw3}. This results in an increase in false matches that only contain sub-phrases, severing the inclusion relation between search terms and query results.

4.5 SECURITY

At rest, the cloud server contains the encrypted documents, EKDi (Di), the conjunctive keyword Bloom filter, BDi, and the n-gram Bloom filters, Bn Di. The security and privacy of the documents are ensured by the symmetric encryption algorithm. The words added to the conjunctive keyword Bloom filter and the n-grams added to the n-gram Bloom filters are hashed with a secret key to prevent the cloud from learning the keywords contained in the documents. The situation is more complex during query. In order to achieve high efficiency, the basic scheme uses the same secret key for the Bloom filters of different documents. As a result, it is possible for the cloud to knowingly verify the existence of an encrypted keyword or n-gram in every document in the corpus. Given enough queries, the cloud could build a statistical distribution of encrypted words. If the cloud has any prior knowledge on the statistics of the corpus, such as that the language is English or that it contains legal documents, it may be able to learn partial information on the data. An intuitive defence against this statistical attack would use different private keys for different documents. However, this would incur significant overhead since filters would have to be computed and verified separately for every document. Instead, we propose a hybrid approach as described in the following section.

4.6 A HYBRID APPROACH AGAINST STATISTICAL ATTACKS

In a typical keyword search scheme, the majority of queries consist of conjunctive keyword searches. Being a specialized search option, phrase searches occur far less frequently.

Therefore, the availability of statistical information for in- dividual keywords would be far greater than that for n- grams. To defend against statistical attacks, the more secure, albeit more expensive, approach of encrypted indexing is used for conjunctive keyword matching, where the statistics of individual keywords are better protected. The approach provides information theoretic security for individual key- words [12] at the cost of having to perform client-side encryption/decryption and to re-encrypt the index when adding files. The use of n-gram Bloom filters for phrase search is retained. In addition to the low availability of statistical information due to the infrequent occurrence of phrase searches, the number of distinct n-grams is also far greater than the number of distinct keywords [24], resulting in a distribution that shows individual probability of occurrence several orders lower than that of keywords [25]. This means it is significantly more difficult to mount a statistical attack against n-grams because far more data is required to recognize the rare occurrences of n-grams while, at the same time, far fewer data is available. Table 1 illustrates this property on our experimental data set. When adding or removing files from the corpus, it should be noted that index update can be delayed to avoid constantly decrypting and re-encrypting the index. That is, the data owner can maintain a small local index which includes recently added and removed files until the next scheduled index update. In the hybrid approach, separate resources are allocated to conjunctive keyword search and phrase search. An encrypted keyword-to-document index, I, is used to support conjunctive keyword search. With the standard setup, two sets of n-gram bloom filters, B2 Di and B3 Di, are used to support phrase search. The encrypted index approach to conjunctive keyword search proceeds as follows. A document collection, D = {D1,D2,...,Dn}, is parsed for a list of keywords, kw_j. An keyword-to-document index, I, is generated mapping key- words to documents such that $I(_{kwj}) = \{d_a,d_b,...,d_n\}$, where $d_i = 1$ if kw_j is linked to the document and $d_i = 0$ otherwise. The resulting index is encrypted and uploaded to the cloud server:

$$I(H_K(kw_j)) = \{E_K(d_a,d_b,...,d_n)\}. \quad (7)$$

To perform a conjunctive keyword search for a set of keywords, $kw_0 = \{kw_1,kw_2,...,kw_q\}$, the data owner computes their hashes, $H_K(kw0)$, using a secret key and sends them to the cloud server. The encrypted index en- tries are returned to the data owner, who computes the intersection of the decrypted index entries and identifies the matching documents:

$$D_K(I(H_K(kw_1))) \,\&\, D_K(I(H_K(kw_2)))\cdots \,\&\, D_K(I(H_K(kw_q))), \quad (8)$$

where & is a bitwise AND operation. If retrieval of the encrypted documents is required, the data owner would then initiate a second round of communication by sending the document identifiers to the cloud server, who would then return the requested documents. The phrase search protocol, which runs independently, in the hybrid construction is identical to that described in section 4.2. Therefore, the response time, communication cost and computational cost associated with phrase search are also identical.

TABLE 1

Average number of distinct n-grams for a sample of 150 documents

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

V.PERFORMANCE ANALYSIS

As outlined in section 4.2, our scheme requires two Bloom filters per document for the purpose of phrase search, one for storing pairs and another for triples. Note that the conjunctive keyword Bloom filters are not required for the purpose of phrase search. The two sets of filters require N(b2+b3) bits of storage on the cloud server, where b2 is the size of a pairs Bloom filter, b3 is the size of a triples Bloom filter and N is the number of documents in the corpus. The data owner needs only to store cryptography keys. Since all existing schemes include conjunctive keyword search capability, we have included the 1-gram filter, b1, in our comparison table 3 and 4. The communication cost of our scheme is similar to [13]. The proposed protocol requires only two messages to be sent, one containing the set bit locations of the query Bloom filter for pairs or triples and the other containing the matched results. Assuming the queried phrase contain $q > 2$ keywords and that k is small, the scheme requires $(q - 2)k\log_2(b_3)$ bits to be sent to the cloud server and $u\log_2(N)$ bits to be sent to the data owner, where u is the number of matched documents. In terms of computation, the scheme requires hashing of triples in the phrase using a secret key, with a total of $3(q - 2)b$ bits, where b is the average number of bits per keyword, and $k(q - 2)$ standard hash computations on the client side to determine the set bit locations of the query filter sent to the server. Upon receiving the query, the server performs a bitwise AND operation between the Bloom filter index entries, each consisting of N bits, for the $k(q - 2)$ set bit location. The matches are then immediately available from the result. The response time of the scheme is dependent on the execution time of the server and the client, and also the transmission time and propagation delay of the messages. The most expensive computation in the protocol is the $3(q-2)b$ keyed hashing performed by the client while non-cryptographic hashing and bit-wise AND are both very efficient operations. In addition, the scheme also enjoys a short transmission time due to a compact description of the query filter and requires the minimal propagation delay of a single round-trip communication. Zittrower's proposal [10] uses an encrypted keyword truncation table that maps distinct keywords with the same truncated ciphertext to different index values. The table allows the data owner to differentiate between entries associated with different keywords but have the same truncated ciphertext in the index table stored on the cloud. Assuming optimal representations, this table requires $x(\log2(x) + b)$ bits, where x is the number of distinct keywords in the corpus and b is the average number of bits per keyword. On the cloud server, two index tables, one mapping

trun- cated encrypted keywords to documents and another to their locations within the documents, are stored. The two tables require $x(12 + p_0 \log_2(N))$ and $x_0(12 + gy)$ bits respectively, where p is the average number of documents associated with a keyword in the corpus and y is the number of bits needed to store a location value. On average, The scheme results in 300 collisions among the encrypted keywords to hide the true search terms. In other words, a query response contains, on average, results belonging to 300 other unrelated keywords, leading to a significant amount of wasted bandwidth and processing. In terms of communication, this implies that up to $300u_i$ log2(N) bits are spent transferring unrelated data during conjunctive keyword search, where ui is the number of candidate documents matched in the search, and 300g(y + log2(x)) + salt) bits are wasted during phrase search for every keyword in the query. When combined with the desired query re- sults, a 12q bits query would yield a response with up to 301ui(log2(N) + 301q(g(y + log2(x)) + salt)) bits of data from the cloud server. Regarding computational cost, the data owner must decrypt all returned entries and, using the encrypted keyword truncation table, identify the results belonging to the searched keywords while discarding collisions. Then, based on the decrypted locations, the data owner identifies documents where the keywords appear in order. Due to the need for client-side encryption and the large amount of false positives processed during search, the scheme has a higher computational requirement than our proposed scheme, where the most expensive operation consists of hash computations. Although the protocol results in only a single round-trip delay to determine matches, it has a long transmission time due to the high number of false positives. If retrieval of matched documents is required, a second round would also be needed by sending the matched identifiers to the cloud server. The response time also degrades due to a high processing time because of a significant amount of decryption operations required at the client which likely possesses much lower computational power than the cloud server. Tang's scheme [11] also uses a table mapping keywords to index values kept locally by the data owner. On the cloud server, a keyword-to-document index and a keyword chain table are stored. While the keyword-to-document index is similar to other existing solutions, the keyword chain table is a structure that allows the verification of keyword chains, combining the use of cryptographic hash and randomly generated location indicators. In order to achieve information theoretic security across all parameters, it was proposed that the keyword chain table to be applied to the entire data set and normalized according to the highest occurring keyword in the document set. In all, x(log2(x) + b) bits of storage is required by the data owner and $x(\log_2(x) + N) + Nx_0(h + d(h + y))$ bits of storage is required by the cloud, where b is the average number of bits per keyword, h is the number of bits used to store a hashed keyword or location value, y is the number of bits to store a location value and d is the number of instance of the most frequent keyword in the corpus. While the client-side storage requirement is low, the keyword chain table, analogous to the location index, is several orders of magnitude greater, due to the normalization to the value of d. In terms of communication, the client sends qlog2(x) bits to the server during the conjunctive keyword search phase and receives q_N bits as results. For matching phrases using the keyword chain table, the data owner sends $u_i(q\log_2(x) + (q - 1)h)$ bits to the cloud server, where ui is the number of candidate documents, and receives $u\log_2(N)$ bits in matching document ID's. While the communication cost of the scheme is an improvement over Zittrower's scheme, communicating index entries and keys is still more costly than the transmission of a single Bloom filter required in our proposed technique. One of the main advantages of Tang's approach is the asymmetric distribution of computational load, where the cloud server, typically possessing significant computational power, performs the majority of the computations. During the conjunctive keyword search phase, the client performs a table lookup and computes the keyed-hash of the queried keywords and sends them to the cloud server. The server then performs a table lookup for the queried entries and return the results to the data owner. Given the candidate documents, the client hashes the keywords under a different secret key in addition to q − 1 chain keys for the chain digests. Upon receiving the hashed phrase and chain keys, the server finds the corresponding entries in the index using binary search, and performs up to d(q − 1) keyed hashes. Due to the size of the keyword chain table, a significant amount of hash computation is required to verify a phrase, especially when a candidate document contains all the keywords but not in consecutive order, where the maximum number of hashes would then be required to reject it as a match. The processing time of the scheme is largely dependent on the server's ability to compute the keyed hashes. The two phase protocol also requires two round-trip delays. Compared to our proposed technique, the scheme has a higher propagation delay and a higher processing time since the hashing algorithm used in Bloom filters is non-cryptographic, which is significantly faster than the cryptographic hash functions used in Tang's scheme. In [12], Poon proposed a solution to address the high communication cost in Zittrower's scheme and the high storage cost noted in Tang's scheme. The central idea was to exploit properties of natural languages to better design the indexes. By considering the almost exponential distribution of words in most languages, it was shown that splitting key- words location entries into pairs dramatically reduces the storage cost of the system. The scheme requires data owners to maintain a dictionary mapping keywords to index values and a list showing the number of times that keywords were split in each file. It was found that the data owner would require x(log2(x)+b) bits for the conjunctive keyword index and 0.27x0N(log2(x0) + log2(k0/2)) bits of storage for the split tables. The cloud server would store the encrypted keyword-to-document index using x(log2(x) + N) bits and the location index tables using 2.5x0N(h + 2y) bits, where h is the number of bits used to represent a hashed keyword and y is the number of bits to store a location value. During the first phase of a phrase query, qlog2(x) bits would be sent to the cloud to identify candidate document and qN bits would be received by the data owner. On average, a query for a single keyword's locations requires 2.5 encrypted keywords to be sent due to splitting. A location query for a random keyword in phrase would then require sending 2.5h bits to the server. Given ui candidates identified in the conjunctive keyword search, 2.5hui bits would need to be sent. In response, the cloud returns the encrypted location entries requiring $2.5ui * 2y = 5uiy$ bits. Hash signatures of the phrase at the identified starting locations are then sent to cloud for matching, requiring $u_{ig}(h + \log_2(N) + y)$ bits, where g is the number of times a keyword appears on average per document. In terms of computation, the cloud server must look up the index entries on each step and perform hash computations of qb bits at uig locations. The data owner must encrypt q keywords for conjunctive keyword search and hash one random keyword for location query. Then, uig encryption and hash computation of q keywords are required to generate the hash signatures for matching. The scheme presents a

significant improvement in terms of practicality over Zittrower and Tang's scheme by offering a much lower storage cost at the cloud and not having to rely on a large number of false positives to maintain security. Phrase search is performed in two rounds of communication, as in Tang's scheme, with the first step identifying candidate documents that contain all keywords. As in Zittrower's scheme, one additional round would be required for retrieval of matched documents if required. In [13], the authors noted that further reduction in storage can be achieved and proposed a scheme based on Bloom filters that focused on minimizing storage cost. For conjunctive keyword search, all distinct keywords in a document is placed in a conjunctive keyword filter to enable keyword-to-document search. For phrase search, keywords are concatenated with their locations and placed inside a keyword location filter to enable location queries. The scheme requires two Bloom filters per document, one for mapping keywords to document and one for determining keyword location. The filters are stored on the cloud server, requiring $N(b_k + b_l)$ bits of storage, where b_k is the size of a conjunctive keyword Bloom filter, $_{bl}$ is the size of a keyword location Bloom filter and N is the number of documents in the corpus. The data owner retains only cryptography keys. In terms of communication cost, the protocol requires that the keyed hash of the keywords for each filter be sent to the cloud server, and the results of the query returned to the data owner. Altogether, the scheme requires 2qh bits to be sent to the cloud server and ulog2(N) bits to be sent to the user, where h is the number of bits per hashed keyword and u is the number of matched documents. Regarding computational cost, the data owner must perform 2q keyed hash computations to generate the trapdoor query. Upon receiving the query, the server performs qk Bloom filter hash computations to produce the query filters. A bitwise AND operation for each conjunctive keyword filter in the document set identifies the candidate documents. Then, for each candidate document, rk Bloom filter hash computation is needed to find the locations of the first word of the phrase, followed by rik hash computation for each additional word to determine matches, where r is the number of keywords in the candidate document and ri is the num- ber of matches for the ith word in the phrase. Generally, r ri. Therefore, approximately uirk hash computations are required during phrase search, where ui is the number of matched documents during conjunctive keyword search. The space-efficiency of Bloom filters allowed the scheme to achieve the lowest storage cost among existing solutions but it required a brute-force approach to identify keyword locations. Although incremental hash functions can improve the verification speed, the computational cost remains high and increases proportionally to the size of the documents. While the scheme requires a single round of communication, the response time of the scheme suffers due to the high processing time required to identify keyword locations. Table 3 shows a comparison of our schemes to existing techniques. For clarity, some terms that do not have a significant impact on the associated cost were omitted. Note that the variants, our scheme (speed) and our scheme (stor- age) are defined in section 6.1 and our scheme (hybrid/*) corresponds to the hybrid approach described in section 4.6. While our scheme (speed) exhibits a high storage cost in order to achieve the fastest processing time by taking full advantage of the Bloom filter index, variants with different values of t, such as our scheme (storage), can achieve fast processing time with a storage cost similar to [12] and 3 times lower than Zittrower's scheme. Regarding the hybrid approach, the technique differs from the scheme in section 4.2 only in its conjunctive keyword search functionality. Its phrase search functionality remains the same. Therefore, its response time, communication cost and computational cost associated with phrase search are identical to our base scheme. The sole difference is the storage cost of the system where resource dedicated to conjunctive keyword search, namely the index, requires higher storage than a 1-gram Bloom filter.

VI. EXPERIMENTAL RESULTS

To compare our results against existing phrase search schemes, we evaluate our algorithm on a corpus consisting of 1500 documents made available by Project Gutenberg [26]. The documents were preprocessed to exclude headers and footers, which include copyright, contact and source information to reduce skewing in the statistics of the data set. Stop words are also omitted. To determine the statistical properties of the corpus and the performance for the various schemes, the Natural Language Toolkit [27] was used. The design of Bloom filters is important to our scheme's performance. In particular, the use of a Bloom filter index requires the filters to be of the same length. Recall that the equation for false positive rate in section 3.1 is as follows:

$$p = (1 - e^{-k\,n/\,m})^k \qquad (9)$$

Figure 3 shows false positive rates between 1% and 10% relative to the number of hash functions and the number of bits needed per entry. A small filter size is preferable in terms of storage. A low false positive rate would reduce communication and computational cost. In particular, using a small number of hash functions greatly improves the execution time since the computational cost is proportional to the number of hash function used. In practice, the number of hash functions, k, needed to minimize false positive rate is rarely used since there is very little improvement in false positive rate as we increase the number of hash functions past a certain threshold. Using a single hash function, k = 1, would reduce the computational cost, but also more than doubles the storage cost to achieve the same false positive rate. The high variance in false positive rate when k = 1 can also be problematic for corpus with high variance in document sizes. As shown in the figure 4, the number of bits per entry and the false positive rate is fairly stable for $k \geqslant 2$ and $m/n \geqslant 10$.

6.1 Selecting the Filter Size

Optimizing for response time requires normalizing the filter size to take advantage of the Bloom filter index. A simple design approach is to ensure that the parameters meet the requirements in the worst case scenario. If an application requires a certain false positive rate, then the filter size can be chosen such that the document with the largest number of distinct keywords or n-grams in the corpus falls within the required rate. In most practical scenarios, it would be the largest document in the corpus. All other smaller documents would exhibit lower than required false positive rate. However, this approach has a high cost in storage. In corpuses where there's a large variance in document sizes, much of the storage is wasted. For example, if we consider the entire Gutenberg corpus of 15620 English documents, the largest document contains 2.8 million words. However, only 88 documents contain more than 140 thousand words and half of the documents in corpus contain less than 20 thousand words. In such scenarios, a trade-off between

response time and storage cost can be made by using t sets of filter sizes where only one of the document set would conform to the largest filter size used, effectively generating t Bloom filter indexes. In the previous example, we can have the largest 88 documents conform to the largest filter size, the following 7722 documents conform to the document containing 140 thousand words and the remaining 7800 documents conform to the document containing 20 thousand words. This simple change would lead to 30 times lower storage requirement than the fastest solution. This approach requires a slight modification of the protocol in section 4.2, where the output of the k hash functions would be sent to the server instead of the set bit locations. To determine the set bit locations, the server simply computes the hash values modulo the filter sizes. The server then proceeds to compute T as usual for each set of filters to determine matches. Note that the limit case of setting t = N, where every document uses exactly the filter size needed to achieve the desired false positive rate, would require the least amount of storage, but would also require the server to verify each filter separately. Nonetheless, it would still achieve much improved response time when compared to existing approaches. We will refer to the approach with t = 1 as "Our scheme (speed)" and t = N as "Our scheme (storage)" in table 3 and 4 to highlight that the former is designed for the highest speed and the latter is designed for the lowest storage cost possible with a trade-off in speed. It should be noted that the best value for t depends on the application and especially on the distribution of file sizes. If all documents have the same number of distinct keywords or n-grams, there is no advantage in storage for choosing t > 1 as all filter sizes would be the same. Having considered various trade-offs, the operating point was heuristically chosen for a false positive rate of p = 5% with k = 2 and m/n = 7.9 for our experimental corpus. A more stringent false positive rate of 1% can be achieved by increasing the number of bits per entry to m/n = 19 while keeping the number of hash functions at k = 2, maintaining a high response time at the cost of more storage. Although we could improve the false positive rate by using more hash functions, increasing the number of hash function from k = 2 to k = 3 would increase our computational cost by 50%, leading to lower performance. The storage requirement is dependent on the number of items that the filters must store. Using the simple ap- proach of normalizing according to the largest document, we would require the number of bits needed to store the number of distinct keywords in the largest document for each conjunctive keyword filter. Similarly, the number of distinct pairs and triples determine the size of each pairs or triples filter. The values of various parameters of the sample Gutenberg document set are listed in table 2. To provide consistent comparison with existing schemes, we consider 1530 documents from the Gutenberg corpus. At 7.9 bits per entry and optimized for speed, a conjunctive keyword Bloom filter would require b1 = x00m/n = 41kB, a pairs fil- ter would require b2 = r2m/n = 2.17mB and a triples filter would require b3 = r3m/n = 2.66mB. Should we wish to minimize storage, we would require b1 = x00m/n = 3.82kB, a pairs filter would require b2 = r2m/n = 23.43kB and a triples filter would require b3 = r3m/n = 28.7kB. Usually, the number of distinct words/pairs/triplets is proportional to the document size. However, in the event of outliers, this could allow an attacker to easily recognize the non-conforming documents. To defend against this, one can design a minimum filter size to file size ratio so that a file with an unusually small number of distinct words would have a filter size larger than what is needed to protect the file's identity. Equivalently, a minimum file size can also be set so that a file requiring an unusually large filter size would be padded to have a correspondingly large file size associated with it.

6.2 Long phrases

Long phrase queries are often used to locate known items rather than to locate resources for a general topic. In many cases, the goal is to identify a single document. Longer phrases also have a very low probability of occurrence and yield fewer matches. Therefore, even with a precision rate of 50%, we would rarely see more than a single false positive for a search query of longer phrases. In our experiment, we never encountered more than a single false positive in queries with phrases containing more than 4 keywords. The small amount of false positives can also be easily identified and removed client-side. As a result, the effect of low precision rate in longer phrases should not have a noticeable detrimental effect in practice. Table 4 summarizes the results of the schemes on the sample Gutenberg document set with the experimental values for the various parameters outlined in table 2. For the hash values of keywords and locations, we assumed that each would require 16 bits in all cases. English words have an average length of 5 letters. Hence, b is set to 40. Since communication and computational costs are query- dependent, related parameters are kept in the formulas and dominant terms were retained for clearer comparisons. In practical scenarios, ui > u > q and Enc(x) ≈ Dec(x) > Hk(x) > Hbf(x) ≈ H(x) > LUT(x) ≈ Mod(x) > And(x),

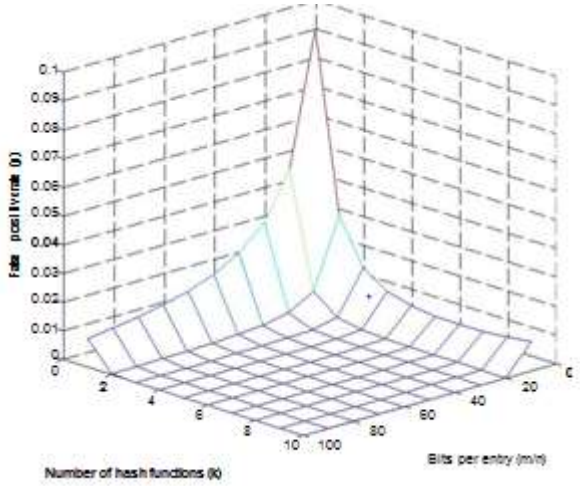

Fig. 3. False positive rate (p) as a function of the number of hash function (k) and bits per entry (m/n)

where Enc(x), Dec(x), Hk(x), Hbf(x), H(x), LUT(x), Mod(x) and And(x) represent respectively the cost of an encryption of x bits, a decryption of x bits, a keyed hash computation of x bits, a Bloom filter hash computation of x bits, a standard hash computation of x bits, a table look up of x elements, the modulus computation of x numerical values and bit-wise AND operation of x bits . Note that the communication and computational cost values for Zit- trower's scheme are worst-case estimates. As shown in the table, our schemes require far lower computational cost on both the data owner and the cloud server. The advantage is particularly notable on the cloud where only basic operations are needed. Our scheme (storage) also achieves almost 5 times lower storage cost than Zittrower's scheme. In terms of communication, our schemes require only a single round trip much like Tang's solution, but requires fewer bits to be sent by either party. It is also interesting to note that the bulk of the communication/computation cost is not dependent on the number of matches for the keywords.

VII CONCLUSION

In this paper, we presented a phrase search scheme based on Bloom filter that is significantly faster than existing approaches, requiring only a single round of communication and Bloom filter verifications. The solution addresses the high computational cost noted in [13] by reformulating phrase search as n-gram verification rather than a location search or a sequential chain verification. Unlike [10], [12], [13], our schemes consider only the existence of a phrase, omitting any information of its location. Unlike [11], our schemes do not require sequential verification, is paralleliz- able and has a practical storage requirement. Our approach is also the first to effectively allow phrase search to run independently without first performing a conjunctive keyword search to identify candidate documents. The technique of constructing a Bloom filter index introduced in section 4.2 enables fast verification of Bloom filters in the same manner as indexing. According to our experiment, it also achieves a lower storage cost than all existing solutions except [13], where a higher computational cost was exchanged in favor of lower storage. While exhibiting similar communication cost to leading existing solutions, the proposed solution can also be adjusted to achieve maximum speed or high speed with a reasonable storage cost depending on the application. An approach is also described to adapt the scheme to defend against inclusion-relation attacks. Various issues on security and efficiency, such as the effect of long phrases and precision rate, were also discussed to support our design choices.

REFERENCES

[1] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in In proceedings of Euro- crypt, 2004, pp. 506–522.

[2] B. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, "Building an encrypted and searchable audit log," in Network and Distributed System Security Symposium, 2004.

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

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

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

[6] 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.

[7] 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.

[8] H. Tuo and M. Wenping, "An effective fuzzy keyword search scheme in cloud computing," in International Conference on Intelligent Networking and Collaborative Systems, 2013, pp. 786–789.

[9] M. Zheng and H. Zhou, "An efficient attack on a fuzzy keyword search scheme over encrypted data," in International Conference on High Performance Computing and Communications and Embedded and Ubiquitous Computing, 2013, pp. 1647–1651.
[10] S. Zittrower and C. C. Zou, "Encrypted phrase searching in the cloud," in IEEE Global Communications Conference, 2012, pp. 764– 770.
[11] Y. Tang, D. Gu, N. Ding, and H. Lu, "Phrase search over encrypted data with symmetric encryption scheme," in International Conference on Distributed Computing Systems Workshops, 2012, pp. 471–480.

TABLE 2
Properties of the sample document set

Average number of documents associated with a keyword, p^0	885.6
Total number of documents, N	1530
Total distinct keywords, x	285396
Average number of keywords per document, r	30364.7
Maximum number of keywords per document, r^0	2815415
Average number of distinct keywords per document, x^0	3959.6
Maximum number of distinct keywords per document, x^{00}	42489
Average number of distinct pairs per document, r2	25177.6
Maximum number of distinct pairs per document, $r2^0$	2252332
Average number of distinct triples per document, r3	30842.5
Maximum number of distinct triples per document, $r3^0$	2759106
Average number of times each keyword appears per document, g	5.6
Number of instance of the most frequent keyword, d	10757
Average number of instance of most frequent word per document, k^0	369.1

TABLE 3
Comparison of phrase search schemes

		Storage	Communication (download)	Computation
Zittrower[10]	data owner	$x(\log_2(x) + b)$	$301u_i(\log_2(N) + 301q(g(y + \log_2(x)) + salt))$	$301u_iDec(301q(g(y+\log_2(x))+salt))+LUT(x)$
	cloud	$x(p^0 \log_2(N) + 12) + N x^0 (12 + gy)$	12q	$LUT(x\ \ x=2^{12}) + u_iLUT(x^0\ \ x^0=2^{12})$
Tang[11]	data owner	$x(\log_2(x) + b)$	$qN + u\log_2(N)$	$LUT(x) + H_k(qb) + u_iH_k(qb) + u_iH_k(b(q\ \ 1))$
	cloud	$x(\log2(x) + N) + N x^0 (h + d(h + y))$	$q\log2(x) + u_i(q\log2(x) + (q\ \ 1)h)$	$LUT(x) + LUT(Nx^0) + d(q\ \ 1)H_k(h)$
Poon[12]	data owner	$x(\log2(x)+b)+0{:}27x^0 N(\log2(x^0)+\log2(k^0=2))$	$qN + 5u_iy$	$qDec(p^0 \log2(N)) + u_i(2{:}5(H_k(\log2(N) + b) + Dec(gy)) + g(Enc(qb) + H(qb + \log2(N) + y)))$
	cloud	$x(\log_2(x) + N) + 2{:}5x^0 N(h + 2y)$	$q\log_2(x) + u_i(2{:}5h + g(h + \log_2(N) + y))$	$LUT(x) + u_iLUT(2{:}5x^0) + u_igH(qb)$
Poon[13]	data owner	0	$u\log_2(N)$	$2H_k(qb)$
	cloud	$N(b_k + b_i)$	2qh	$kH_{bf}(qb) + uirkH_{bf}(b + \log_2(r))$
Our scheme (speed)	data owner	0	$u\log_2(N)$	$H_k(3(q\ \ 2)b) + k(q\ \ 2)H_{bf}(16)$
	cloud	$N(b_1 + b_2 + b_3)$	$(q\ \ 2)k\log_2(b_3)$	$And(k(q\ \ 2)N)$
Our scheme (storage)	data owner	0	$u\log_2(N)$	$H_k(3(q\ \ 2)b) + k(q\ \ 2)H_{bf}(16)$
	cloud	$N(b_1 + b_2 + b_3)$	$(q\ \ 2)k\log_2(b_3)$	$Mod(k(q\ \ 2)N) + And(Nb_3)$
Our scheme (hybrid/speed)	data owner	$x(\log_2(x) + b)$	$u\log_2(N)$	$H_k(3(q\ \ 2)b) + k(q\ \ 2)H_{bf}(16)$
	cloud	$x(\log_2(x) + N) + N(b_2 + b_3)$	$(q\ \ 2)k\log_2(b_3)$	$And(k(q\ \ 2)N)$
Our scheme (hybrid/storage)	data owner	$x(\log_2(x) + b)$	$u\log_2(N)$	$H_k(3(q\ \ 2)b) + k(q\ \ 2)H_{bf}(16)$
	cloud	$x(\log_2(x) + N) + N(b_2 + b_3)$	$(q\ \ 2)k\log_2(b_3)$	$Mod(k(q\ \ 2)N) + And(Nb_3)$

TABLE 4

Comparison of phrase search schemes for a sample of 1500 documents

	Zittrower[10]		Tang[11]		Poon[12]		Poon[13]	
	data owner	cloud	data owner	cloud	data owner	cloud	data owner	cloud
Storage	1:98MB	392:5MB	1:98MB	242:8GB	5:78MB	139:3MB	0MB	49:47MB
Communication	$_i$ kb	bits	bits	$_i$ bits	bits	bits	bits	$_4$ bits
Computation	17 10 $u_iDec(q)$	$u_iLUT(3959)$	$u_iH_k(40(2q\ \ 1))$	10 $(q\ \ 1)H_k(16)$	$5{:}6u_iEnc(40q) + Dec(9369q)$	$5{:}6u_iH(40q)$	$2H_k(40q)$	6 10 $u_iH_{bf}(55)$
	Our scheme (speed)		Our scheme (storage)		Our scheme (hybrid/speed)		Our scheme (hybrid/storage)	
	data owner	cloud	data owner	cloud	data owner	cloud	data owner	cloud
Storage	0MB	7:28GB	0MB	85:6MB	1:98MB	7:27GB	1:98MB	130:55MB

Communication	10:6u bits	44(q_2) bits	10:6u bits	30(q_2) bits	10:6u bits	44(q_2) bits	10:6u bits	30(q_2) bits
Computation	$H_K(120(q\_2))$	And(3060(q_2)))	$H_K(120(q\_2))$	Mod(3060(q_2))	$H_K(120(q\_2))$	And(3060(q_2)))	$H_K(120(q\_2))$	Mod(3060(q_2))

[12] H. Poon and A. Miri, "An efficient conjunctive keyword and phrase search scheme for encrypted cloud storage systems," in IEEE International Conference on Cloud Computing, 2015.
[13] ——, "A low storage phrase search scheme based on bloom filters for encrypted cloud services," to appear in IEEE International Conference on Cyber Security and Cloud Computing, 2015.
[14] H. S. Rhee, I. R. Jeong, J. W. Byun, and D. H. Lee, "Difference set attacks on conjunctive keyword search schemes," in Proceedings of the Third VLDB International Conference on Secure Data Management, 2006, pp. 64–74.
[15] K. Cai, C. Hong, M. Zhang, D. Feng, and Z. Lv, "A secure conjunctive keywords search over encrypted cloud data against inclusion-relation attack," in IEEE International Conference on Cloud Computing Technology and Science, 2013, pp. 339–346. [16] Y. Yang, H. Lu, and J. Weng, "Multi-user private keyword search for cloud computing," in IEEE Third International Conference on Cloud Computing Technology and Science, 2011, pp. 264–271.
[17] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked key- word search over encrypted cloud data," in International Conference on Distributed Computing Systems, 2010, pp. 253–262.
[18] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamas- sia, "Practical oblivious storage," in Proceedings of the Second ACM Conference on Data and Application Security and Privacy, 2012, pp. 13–24.
[19] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in Proceedings of the 36th Annual Symposium on Foundations of Computer Science, 1995, pp. 41–50.
[20] S. Ruj, M. Stojmenovic, and A. Nayak, "Privacy preserving access control with authentication for securing data in clouds," in Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012, pp. 556–563. [21] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in Proceedings of the 2000 IEEE Symposium on Security and Privacy, 2000, pp. 44–55.
[22] E.-J. Goh, "Secure indexes," Cryptology ePrint Archive, Report 2003/216, 2003.
[23] C. Liu, L. Zhu, L. Li, and Y. Tan, "Fuzzy keyword search on encrypted cloud storage data with small index," in 2011 IEEE International Conference on Cloud Computing and Intelligence Systems, 2011, pp. 269–273.
[24] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," Computational Linguistics, vol. 18, no. 4, pp. 467–479, 1992.
[25] D. Jurafsky and J. H. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. Prentice Hall, 2009.
[26] "Project Gutenberg," https://www.gutenberg.org/wiki/Main Page, accessed: 2014.
[27] E. L. Bird, Steven and E. Klein, Natural Language Processing with Python. O'Reilly Media Inc, 2009.