A Low Storage Phase Search Scheme based on Bloom Filters for Encrypted Cloud Services

Hoi Ting Poon and Ali Miri
Department of Computer Science
Ryerson University
Toronto, Ontario, Canada
{*hoiting.poon, ali.miri@ryerson.ca*}

***Abstract*—Despite the many benefits of cloud technologies, there have also been significant concerns regarding its security and privacy. To address the issues, much effort have been made towards development of an encrypted cloud system. One of the key features being investigated is the ability to search over encrypted data. Although many have proposed solutions for conjunctive keyword search, few have considered phrase searching techniques over encrypted data. Due to the increased amount of information required to identify phrases, existing phrase search algorithms require significantly more storage than conjunctive keyword search schemes. In this paper, we propose a phrase search scheme, which takes advantage of the space efficiency of Bloom filters, for applications requiring a low storage cost. It makes use of symmetric encryption, which provides computational and storage efficiency over schemes based on public key encryption. The scheme provides simple ranking capability, can be adapted to non-keyword search and is suitable against inclusion-relation attack.**

***Keywords*-Phrase search, Bloom filters, Privacy, Security, Encryption.**

I. Introduction

In recent years, cloud technologies have garnered increasing interest for providing reliable, scalable and accessible computing services. While many have embraced its advantages, others are hesitant amid security and privacy concerns. Many valuable applications such as in health and financial services require handling of confidential documents, which can not be stored in plain on servers controlled by cloud operators. To address the issue, encryption is generally agreed to be needed. However, many functions available in storage systems and databases, such as search, deduplication and auditing, do not translate intuitively to encrypted data. In particular, the ability to search has been identified as one of the most important features needed in encrypted storage systems and is actively investigated by researchers.

Boneh [1] was one of the first researchers to address the issue, proposing a public key encryption scheme which allows a set of keywords to be searchable without revealing the content of emails. Waters [2] examined the issue of searching over encrypted audit logs. While early works have focused on single keyword searches, more recent works considered searches involving multiple keywords [3][4], termed conjunctive keyword search. Features such as ranking of search results have also been looked at in [5]. Meanwhile, Tuo [6] and Zheng [7] devised search schemes which would return relevant results despite potentially erroneous keywords. The ability to search for phrases were also recently investigated [8][9].

In this paper, we present a low storage solution for encrypted phrase search. The scheme is capable of basic ranking and can be adapted to search for non-indexed keywords and defend against Inclusion-Relation attacks [10]. We begin by describing two leading phrase search algorithms and the properties of Bloom filters which form the basis of our solution in II. The conjunctive keyword and phrase search protocols are presented in section III, along with explanations on various features and design choices. A detailed performance analysis can be found in section IV. Finally, experimental results is given in section V.

II. Background

Performing phrase search in encrypted data was only recently investigated by researchers. The key difference between conjunctive keyword search and phrase search is that, in addition to containing the queried keywords, they must also appear contiguously in the specified order in the document. Therefore, a phrase search scheme must have some knowledge on the locations of the keywords.

Zittrower [8] was the first to address the problem. He proposed a solution using a standard keyword-to-document index and a keyword location index. The keyword-to-document index maps keywords to the associated documents. The location index contains the position of the keywords within each document. All keywords and entries are encrypted. It was noted that the keyword and location indexes are particularly susceptible to statistical attacks. Since certain words are more common than others in every natural language, the distribution of keywords in the indexes could reveal the keywords themselves. To defend against statistical attack, truncation of encrypted keywords is used to generate false positives in query results to hide the true search terms. As a result, part of the index must be stored

978-1-4673-9300-3/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCloud.2015.30

client-side and the search is also performed by the client rather than the cloud.

Tang [9] addresses the privacy concerns by proposing a solution with provable security using normalization. The technique also uses two index tables: a keyword-to-document index and a keyword chain table. Rather than storing the locations of individual keywords, Tang opted to use a table to verify existence of pairs of keywords. To achieve security against statistical attacks, the keyword chain table is normalized against all documents in the corpus by filling in random data so that every entry contains the same number of elements as the largest entry prior to normalization, resulting in a uniform distribution of entries in the table. The high level of security achieved, however, comes at a high storage cost as the index table requires significant storage, which hinders its practicality.

A. Bloom filters

The use of Bloom filters has been suggested for simple and conjunctive keyword search [11][10] to reduce storage cost and provide keyword security in the form of false positives. However, their usability in phrase search has not been explored. In [10], Cai proposed using Bloom filters to generate false positives by querying only a subset of the search terms. In [12], a phonetic coding system is used prior to Bloom filter generation to allow for approximate searching and case-sensitive search. Bellovin also described [13] a solution for secure multi-party data sharing. However, the scheme uses public key encryption and can be computationally expensive for certain applications.

Bloom filters are space-efficient probabilistic data structure used to test whether an element is a member of a set. A Bloom filter is an array of m-bits and k hash functions, $H_i(x)$, are used to map to the m-bits in the filter. Initially, the Bloom filter is set to all zeros. To add an element, a, to the filter, we compute $H_i(a)$ for $i = 1$ to k, and set the corresponding positions in the filter to 1.

While Bloom filters are space-efficient, it can falsely identify an element as member of a set. False negatives, on the other hand, are impossible. Given k hash functions, n items inserted and m bits used in the filter, the probability of false positives is approximately $p = (1 - e^{-kn/m})^k$ and the following equality minimizes the false positive rate: $k = \frac{m}{n} ln2$. In most applications, it is desirable to minimize false positives. In keyword search schemes, however, the ability to generate false positives provides a way to perform privacy-protected searching by hiding the search terms.

B. Inclusion-Relation attacks

In [10], Cai described the notion of inclusion-relation (IR) attacks, which states that two query sets, a and b, where a is the subset of b, would imply b includes the set of keywords from a. If a cloud server has some knowledge of the statistical properties of the incoming search terms, it can potentially discover some of the keywords.

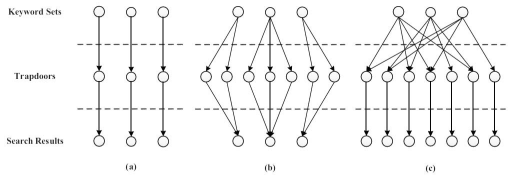

Figure 1. Relationship between keyword set, trapdoor and search result[10]

An interesting property of the proposed technique is that it can be adapted to defend against such attacks by ensuring that a set of keywords can lead to many possible queries (trapdoors) due to the inclusion of false positives. Also, since different sets of keywords can lead to the same positions in the Bloom filter being set to 1, different keyword sets can also lead to the same query. This provides some level of privacy. Figure 1 shows the different mapping of keywords to trapdoors and search results between various conjunctive keyword search schemes. Our proposed technique is of type C, which was suggested to provide the best defense against inclusion-relation attacks [10].

III. Phrase search scheme based on Bloom filters

A Bloom filter is a space efficient data structure used to test whether an element is a member of a set. In a search scheme, it can be used to test whether a keyword is associated with a document. The data structure can also be adopted for our phrase search scheme. A traditional phrase search scheme, as in [8][9], uses a keyword-to-document index and a location/chain index to map keywords to documents and match phrases. We describe the use of Bloom filters to support the functionality of these indexes to further reduce the storage cost.

A. Conjunctive keyword search protocol

To provide conjunctive keyword search capability, each document, D_i, is parsed for a list of keywords kw_j. A Bloom filter of size m is initialized to contain all zeros. Each keyword is hashed using a private key to produce $H_{k_c}(kw_j)$ before passing into k Bloom filter hash functions and the result is used to set k bits in the Bloom filter. This addition of keywords as members results in a Bloom filter for each document: $B_{D_i} = \{b_1, b_2, ...b_m\}$ where $b_i \in \{0, 1\}$. The document collection, $D = \{D_1, D_2, \ldots, D_n\}$, is then encrypted and uploaded along with the Bloom filters to the cloud server. Note that each Bloom filter can be viewed as the equivalent of an entry in an index table.

To perform a conjunctive keyword search, the user sends a set of keywords $kw' = \{kw_1, kw_2...kw_q\}$ to the data owner. The data owner computes a keyed hash, $H_{k_c}(kw')$, of the

encrypted keywords, randomizes their order, and sends them to the cloud server. Upon receipt, the cloud server computes a query Bloom filter, $T = \{t_1, t_2, ...t_m\}$, for $H_{k_c}(kw')$ and verifies the keywords by comparing against the Bloom filters for each document. A match is found if $T = T\&B_{D_i}$, where $\&$ denotes a bitwise *and* operation. Finally, the cloud server sends the matched documents to the user.

B. Phrase search protocol

To provide phrase search capability, each keyword is concatenated with its location, $H_{k_l}(kw_j)|j$, and passed into k hash functions and the result is used to set k bits in the Bloom filter, Bl_{D_i}. The resulting keyword location filter is stored alongside the conjunctive keyword filter on the cloud. To perform a phrase search, the user performs a conjunctive keyword search for all the keywords in the phrase as previously described, except the user also sends the hash of the phrase under a different private key,

$$H_{k_l}(kw') = \{H_{k_l}(kw_1), H_{k_l}(kw_2)...H_{k_l}(kw_q)\}, \qquad (1)$$

to the cloud. For each document containing all the keywords, the location of the first keyword in the phrase is queried. To do so, the cloud server verifies whether $H_{k_l}(kw_1)|s$ is a member of Bl_{D_i}, for $s = 1$ to r', where r' is the size of the candidate document. Then, for each $H_{k_l}(kw_1)|s'$ found to be a member of Bl_{D_i}, the cloud verifies whether $H_{k_l}(kw_2)|s'+1$ is a member of the set to determine whether the next keyword is in sequence. The cloud then proceeds similarly to determine whether the keywords in the phrase follows in proper order by verifying the membership of $H_{k_l}(kw_{2+i})|s' + i + 1$ for each $H_{k_l}(kw_{1+i})|s' + i$ found, narrowing the search at each step. The documents where phrases were found are returned, that is the set of D_i where

$$\{H_{k_l}(kw_{1+j})|s' + j\} \in Bl_{D_i} \text{ and } j = 1 \text{ to } q. \qquad (2)$$

Note that the phrase search requires only 2 exchanges with the cloud server. The data owner relays the query Bloom filter T and the encrypted phrase $H_{k_l}(kw')$ to the cloud. The cloud performs the conjunctive keyword search and location search locally and returns the results.

Despite its advantage in reducing storage, Bloom filter does have some drawbacks in security and computational cost. Its use leads to a probabilistic generation of false positives to hide the search terms. Using a larger filter leads to lower false positive rate, but also lowers protection of search terms and higher storage cost. Using a small filter leads to greater protection, low storage cost but larger false positives and computational/communication cost from searching for non-relevant terms. Aside from the trade-off in security, the scheme can require a higher computational cost than an indexing approach, due to the iterative hash computations during phrase search to verify the keyword-location membership. The choice of hash function is therefore an important factor in the performance of the scheme.

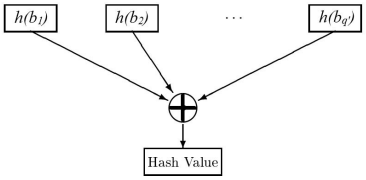

Figure 2. Incremental hash function for Bloom filter

C. Incremental hash functions

While popular cryptographic hash functions such as SHA2 and MD5 are often cited, they are less suitable for Bloom filter constructions. Since keywords are encrypted prior to addition into the filter to hide the search terms from the cloud, security properties such as preimage and collision resistance are not needed for our application. Instead, speed and low collision rate are valued.

From section III-B, it can be observed that $H_{k_l}(kw_1)$ does not change as we iterate through different values of s when computing $H_i(H_{k_l}(kw_1)|s)$. Therefore, a hash function that can efficiently compute $H_i(x_1|x_2|x_3')$ given $H_i(x_1|x_2|x_3)$ can greatly improve performance. These are generally called incremental hash functions introduced by Bellare [14]. A series of efforts into their development culminated in a randomize-then-combine construction [15], in which standard cryptographic hash functions such as SHA2 and MD5 can be made incremental. Since the hash functions in the Bloom filter are not security-critical, we simplify the construction to increase performance. The resulting design is shown in figure 2, where $h()$ is a standard hash function and b_i are input blocks. The key features being a simple XOR combine function and a parallel design for fast computation.

While the randomize-then-combine construction by Bellare holds a slight advantage in performance compared to the more popular Merkle-Damgård construction. The latter can also be used to perform incremental hashing provided that the location is attached at the end since the Merkle-Damgård process is iterative rather than parallel.

For the hash function, $h()$, MurmurHash3a is currently among the fastest non-cryptographic hash functions available while providing randomness and collision properties similar to its cryptographic counterparts [16].

D. Non-keyword search

Note that our scheme does not restrict the encryption algorithm used for the document set. However, its choice could enable certain features such as the ability to perform searches on non-keywords, i.e. words that are not added to the Bloom filters. Common words in a language, such as "the", "a", "is", also called stop words, are often omitted since they are functional terms of little relevance to the document's content. While this improves performance, it

also limits the ability to search for phrases that include such terms, as is the case in [8][9].

To enable non-keyword search, we adapt a technique demonstrated in [17] to our scheme. Briefly, we encrypt the document set using symmetric encryption, such as AES, running in counter mode with the initialization vector set to $H(D_i)$. A list of the 500 most common words in order of frequency in the language of the document set excluding stop words is maintained by the data owner. To perform a phrase search, a conjunctive keyword search is performed with all the keywords in the phrase as in section III-A. Then, the least common word in the phrase is identified using the frequency list and its location is queried by verifying whether $H_{k_l}(kw_1)|s$ is a member of Bl_{D_i} for each candidate document. Given the locations, the owner returns

$$\{H(E_{K_{D_i},j_s}(kw_1, kw_2, \ldots, kw_q)), i, j_s\} \tag{3}$$

where i is the index of the matched document, j_s is the identified starting location of the phrase and $H()$ is a non-cryptographic hash function, for each match. $E_{K_{D_i,j_s}}()$ represents the symmetric encryption of the phrase at location j_s of document D_i. The cloud then computes $H(E(w_{j_s}, w_{j_s+1}, \ldots, w_{j_s+q}))$, where $E(w_j)$ is the j^{th} stored word in document i. Matched phrases are found where the following equality holds:

$$H(E(w_{j_s}, \ldots, w_{j_s+q})) = H(E_{K_{D_i,j_s}}(kw_1, \ldots, kw_q)). \tag{4}$$

E. Ranking and adding/removing documents

Our scheme can also rank the query results by tracking the number of matched phrases per document. Proximity ranking, suggested by Zittrower [8], can also be used since the location of the keywords can be queried.

Another advantage of this approach is the ease in adding and removing documents since each document retains its own pair of Bloom filters. In the algorithm proposed by Tang [9], a problem could arise where a new document's most common term appears more frequently than the most common term in the document set. As a result, it could not be added without regenerating the index.

F. Modified phrase search scheme against IR attacks

An inclusion-relation attack can be carried out against the basic scheme if an attacker gains access to a significant amount of known queries and their associated search results. The class of searching algorithms of type C noted in figure 1 was proposed to defend against such attacks.

To adapt our basic scheme in section III to defend against IR attacks, we increase the false positive rate by randomly removing z terms from the beginning and the end of the phrase being queried. Due to the uniqueness of long phrases, more terms can be removed to generate false positives. For our experimental data set, $z = \lfloor q/3 \rfloor$ was found to be effective for $q \leq 6$ and $z = q - 3$ for $q > 6$. For example, a query phrase, $kw' = \{kw_1, kw_2, kw_3\}$, would be queried randomly as $kw' = \{kw_1, kw_2\}$ or $kw' = \{kw_2, kw_3\}$. This results in false matches that contain only sub-phrases, severing the inclusion relation between search terms and query results for queries with common terms.

G. Security

The cloud server contains the encrypted documents, $E_{K_{D_i}}(D_i)$, the conjunctive keyword Bloom filter, B_{D_i}, and the location Bloom filter, Bl_{D_i}. The security and privacy of the documents are ensured by the symmetric encryption algorithm. The filters do not reveal meaningful information since all members are meshed together in the structure. The words added to the conjunctive keyword Bloom filter are encrypted to prevent the cloud from learning the keywords that are contained in the documents. The location Bloom filter operates similarly, but the keywords are encrypted with a different key to sever the link between the two filters. If the same encryption key had been used, the cloud would be able to perform a location query even when it's not requested.

IV. Performance Analysis

As outlined in section III-B, our scheme requires two Bloom filters per document, one for mapping keywords to document and one for determining keyword location. The two sets of filters require $N(b_k + b_l)$ bits of storage on the cloud server, where b_k is the size of a conjunctive keyword Bloom filter, b_l is the size of a keyword location Bloom filter and N is the number of documents in the corpus. The data owner needs only to store cryptography keys. The communication cost also compares favorably with existing schemes. The proposed protocol requires only two messages to be sent, one containing the keyed hash of the keywords for each filter, and the other containing the results of the query. Altogether, the scheme requires $2qh$ bits to be sent to the cloud server and $ulog_2(N)$ bits sent to the user, where h is the number of bits per hashed keyword and u is the number of matched documents. In terms of computation, the scheme requires $2q$ keyed hash computations on the client side to generate the trapdoor query sent to the server. Upon receiving the query, the server performs qk Bloom filter hash computations to produce the query filters, followed by a bitwise AND operation for each conjunctive keyword filter, B_{D_i}, in the document set. Then, for each matched document, rk Bloom filter hash computation is needed to locate the first word in the phrase, followed by r_ik hash computation for each additional word in the phrase, where r is the number of keywords in the candidate document and r_i is the number of matches for the i^{th} word in the phrase. In most practical scenarios, $r \gg r_i$, resulting in approximately u_irk hash computations required during phrase search, where u_i is the number of matched documents during conjunctive keyword search. The response time of the scheme is dependent on the execution time/computational power of the server and

the client, and also the transmission and propagation time of the messages. While the computational load will likely be the more important factor, our scheme also has the advantage of a short transmission time due to a small query filter and requiring only a single round-trip delay time to complete. Therefore, the distance between the server and client has a lower impact on the performance of the system.

Zittrower's truncated ciphertext approach requires a client-side dictionary mapping distinct keywords to index values to differentiate between entries associated with different keywords under the same key in the index table [8]. Assuming optimal representations, this table requires $x(log_2(x) + b)$ bits of storage, where x is the number of distinct keywords in the corpus and b is the average number of bits per keyword. On the server, two index tables are stored: one mapping truncated encrypted keywords to documents and another to their locations within the documents. The two tables require $x(12 + p' \, log_2(N))$ and $x'(12 + gy)$ 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. The scheme relies on false positives to provide security, with an average of 300 collisions among the encrypted keywords. Hence, a query would also return results belonging to 300 other unrelated keywords, leading to a significant amount of wasted bandwidth and processing. In terms of communication, this results in up to $300u_i \, log_2(N)$ bits wasted during conjunctive keyword search, where u_i is the number of candidate documents matched in the search, and $300g(y + log_2(x)) + salt)$ bits wasted during phrase search for every keyword in the query. Together with the true results, the scheme would respond to a client's $12q$ bits query with up to $301u_i(log_2(N) + 301q(g(y + log_2(x)) + salt))$ bits of data. In terms of computation, the client must first decrypt all the returned entries and then use the dictionary to identify the relevant results while discarding collision. Further processing follows to identify keywords that are in order based on the decrypted locations. The computational load is naturally higher compared to the non-cryptographic hashing used in the proposed scheme and must be carried out by the client. Although the scheme requires only a single round-trip delay, the transmission time is relatively long due to the high number of false positives. The response time also suffers due to the significant amount of decryption operations. Since the client likely possesses much lower computational power than the server, it would also require a longer execution time to complete the decryptions.

Tang's scheme [9] similarly stores a keyword dictionary on the client while a keyword-to-document index and keyword chain tables are stored on the cloud server, requiring $x(log_2(x) + b)$ bits of client-side storage and $x(log_2(x) + N) + Nx'(h + d(h + y))$ bits of server-side storage, where b is the average number of bits per keyword, h is the number of bits 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. Although the storage requirement for the client and the keyword-to-document index is similar to Zittrower's scheme, 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 $qlog_2(x)$ bits to the server during the conjunctive keyword search and receives qN bits as results. For phrase matching, the client sends $u_i(qlog_2(x)+(q-1)h)$ bits to the server, where u_i is the number of candidate documents, and receives $ulog_2(N)$ bits in matching document ID's. Although its communication cost is lower than Zittrower's scheme, communicating index entries and keys still costs more than the transmission of a single Bloom filter and the encrypted key terms required by the proposed technique. One of the main advantages of Tang's approach is in allowing the cloud to perform the majority of the computations. During conjunctive keyword search, the client performs a dictionary lookup and computes the keyed-hash of the keywords and sends them to the server. The server similarly performs a table lookup for the queried entries. During phrase search, the client hashes the keywords under a different private 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 via 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 response time of the scheme is then largely dependent on the server's ability to compute the keyed hashes. When compared to the proposed scheme, it's important to note that the hashing algorithm used in Bloom filters is non-cryptographic and incremental, which is significantly faster than the cryptographic hash functions used in Tang's scheme. It should also be noted that the scheme requires two round-trip delays compared to the one required in our scheme, with a similar transmission time.

V. Experimental Results

We evaluate the proposed algorithm on a corpus consisting of 1500 documents made available by Project Gutenberg [18] and compared our results against existing phrase search schemes. 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 schemes, the Natural Language Toolkit [19] was used.

As Bloom filter forms the basis of our approach, its design parameters are an important factor on the scheme's performance. We rewrite the equation for false positive rate

in section II-A as follows:

$$\frac{m}{n} = \frac{-k}{ln\left(1-e^{\frac{ln(p)}{k}}\right)}. \quad (5)$$

Figure 3 shows the number of bits needed per entry to achieve false positive rates between 1% and 10% depending on the number of hash functions used. To reduce storage cost, a small filter is desirable. A low false positive rate would be beneficial in terms of communication and computational cost. Most importantly, 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. Note that the the optimal k for minimizing false positive rate is rarely used in practice since there is often very little improvement in false positive rate as we increase the number of hash functions past a certain threshold. While the limit case of using a single hash function, $k = 1$, would minimize the computational cost, it also more than doubles the storage cost compared to using more hash functions. As shown in the figure, the number of bits per entry and the false positive rate is fairly stable for $k \geq 2$. Although we could improve the false positive rate by using more hash functions, increasing the number of hash function used from $k = 2$ to $k = 3$ would increase our computational cost by 50% while reducing storage cost by no more than 25%. Having considered the various trade-offs, the operating point was heuristically chosen for a false positive rate of $p = 5\%$ with $k = 2$ and $m/n = 7.9$. To determine the number of bits needed for the Bloom filters, B_{D_i} and Bl_{D_i}, we need the number of items that each filter must store. That is the total number of keywords per document and the number of distinct keywords per document. Their values among various parameters of the sample Gutenberg document set are listed in table I. At 7.9 bits per entry, a conjunctive keyword Bloom filter would require, on average, $b_k = mx'/n = 3.82$kB and a location filter would require $b_l = mr/n = 29.286$kB.

Table III summarize the results of the schemes on the sample Gutenberg document set with the experimental values for the various parameters outlined in table I. The hash values of keywords and locations are assumed to require 16 bits. Since communication and computational costs are query-dependent, related parameters are kept in the formulas and approximations were made to retain dominant terms for clearer comparisons. In practical scenarios, $u_i > u > q$ and $Dec(x) > H_k(x) > H_{bf}(x) > LUT(x)$, where $Dec(x)$, $H_k(x)$, $H_{bf}(x)$ and $LUT(x)$ represent the cost of a decryption of x bits, a keyed hash computation of x bits, a Bloom filter hash computation of x bits and a table look up of x elements. Note that the communication and computational cost values for Zittrower's scheme are worst-case estimates. As shown in the table, our scheme requires almost 8 times lower storage cost than Zittrower's scheme. Aside from the low storage cost, it also achieves a lower communication cost and would vastly outperform both schemes when there are many candidate documents. As for computational cost, our scheme requires more work to be performed by the cloud server, but the load is far lighter on the client, which is desirable in a cloud based solution.

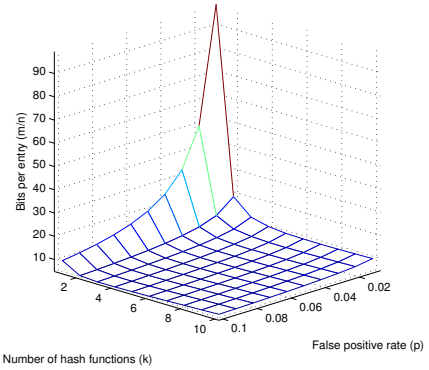

Figure 3. Bits per entry (m/n) as a function of the number of hash function (k) and false positive rate (p)

VI. Conclusion

In this paper, we presented a phrase search scheme based on Bloom filter that achieves 8 times lower storage cost in our experiment than the existing solutions while exhibiting similar or better communication and computational requirements. The proposed solution provides basic ranking capability, can be adapted to non-keyword search and is suitable against inclusion-relation attacks. We discussed various issues on security and efficiency that supported our design choices. Furthermore, a detailed analysis on the performance of several existing phrase search schemes was provided, along with formulas to evaluate their performance based on the characteristics of corpus.

References

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

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

[3] Maozhen Ding, Fei Gao, Zhengping Jin, and Hua 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 unstructured text," in *International Conference on Network and System Security*, 2011, pp. 285–289.

Table I
PROPERTIES OF THE SAMPLE DOCUMENT SET

Average number of documents associated with a keyword, p'	885.6
Total number of documents, N	1530
Total distinct keywords, x	285396
Average number of keywords per document, r	30364.7
Average number of distinct keywords per document, x'	3959.6
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'	369.1

Table II
COMPARISON OF PHRASE SEARCH SCHEMES

		Storage	Communication (download)	Computation
Zittrower [8]	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	$x(p'log_2(N)+12)+Nx'(12+gy)$	$12q$	$LUT(x-x/2^{12})+u_iLUT(x'-x'/2^{12})$
Tang [9]	data owner	$x(log_2(x)+b)$	$qN+ulog_2(N)$	$LUT(x)+H_k(qb)+u_iH_k(qb)+u_iH_k(b(q-1))$
	cloud	$x(log_2(x)+N)+Nx'(h+d(h+y))$	$qlog_2(x)+u_i(qlog_2(x)+(q-1)h)$	$LUT(x)+LUT(Nx')+d(q-1)H_k(h)$
Our scheme	data owner	0	$ulog_2(N)$	$2H_k(qb)$
	cloud	$N(b_k+b_l)$	$2qh$	$kH_{bf}(qb)+u_irkH_{bf}(b+log_2(r))$

Table III
COMPARISON OF PHRASE SEARCH SCHEMES FOR A SAMPLE OF 1500 DOCUMENTS

	Zittrower [8]		Tang [9]		Our scheme	
	data owner	cloud	data owner	cloud	data owner	cloud
Storage	1.14MB	374.3MB	1.14MB	226.1GB	0MB	47.6MB
Communication	$16740.5u_iq$ kb	$12q$ bits	$10.6u+1530q$ bits	$(34q-16)u_i$ bits	$10.6u$ bits	$32q$ bits
Computation	$17*10^6u_iDec(q)$	$u_iLUT(3959)$	$u_iH_k(40(2q-1))$	$10^4(q-1)H_k(16)$	$2H_k(5q)$	$6*10^4u_iH_{bf}(55)$

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

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

[7] Minghui Zheng and Huihua 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.

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

[9] Yinqi Tang, Dawu Gu, Ning Ding, and Haining Lu, "Phrase search over encrypted data with symmetric encryption scheme," in *International Conference on Distributed Computing Systems Workshops*, 2012, pp. 471–480.

[10] Ke Cai, Cheng Hong, Min Zhang, Dengguo Feng, and Zhiquan 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.

[11] Eu-Jin Goh, "Secure indexes," Cryptology ePrint Archive, Report 2003/216, 2003.

[12] S.K. Pal, P. Sardana, and A. Sardana, "Efficient search on encrypted data using bloom filter," in *International Conference on Computing for Sustainable Global Development*, 2014, pp. 412–416.

[13] Steven M. Bellovin and William R. Cheswick, "Privacy-enhanced searches using encrypted bloom filters," 2004.

[14] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing," in *International Cryptology Conference Santa Barbara*, 1994, pp. 216–233.

[15] M. Bellare and D. Micciancio, "A new paradigm for collision-free hashing: Incrementality at reduced cost," in *International Conference on the Theory and Application of Cryptographic Techniques*, 1997, pp. 163–192.

[16] "Smhasher & murmurhash," https://code.google.com/p/smhasher/, Accessed: 2015-04-30.

[17] H.T. Poon and A. Miri, "An efficient conjunctive keyword and phase search scheme for encrypted cloud storage systems," to appear in *IEEE International Conference on Cloud Computing*, 2015.

[18] "Project Gutenberg," https://www.gutenberg.org/wiki/Main_Page, Accessed: 2014.

[19] Edward Loper Bird, Steven and Ewan Klein, *Natural Language Processing with Python*, O'Reilly Media Inc, 2009.