An Efficient Conjunctive Keyword and Phase Search Scheme for Encrypted Cloud Storage Systems

Hoi Ting Poon and Ali Miri
Department of Computer Science
Ryerson University
Toronto, Ontario, Canada
hoiting.poon@ryerson.ca, samiri@scs.ryerson.ca

***Abstract*—There have been increasing interest in the area of privacy-protected searching as industries continue to adopt cloud technologies. Much of the recent efforts have been towards incorporating more advanced searching techniques. Although many have proposed solutions for conjunctive keyword search, it is only recently that researchers began exploring phrase search over encrypted data. In this paper, we present a scheme that incorporates both functionalities. Our solution makes use of symmetric encryption, which provides computational and storage efficiency over schemes based on public key encryption. By considering the statistical properties of natural languages, we were able to design indexes that significantly reduce storage cost when compared to existing solutions. Our solution allows for simple ranking of results and requires a low storage cost while providing document and keyword security. By using both the index and the encrypted documents to performs searches, our scheme is also currently the only phrase search scheme capable of searching for non-indexed keywords.**

***Keywords*-Conjunctive keyword search, Phrase search, Privacy, Security, Encryption.**

I. Introduction

The advent of cloud technologies promises greater scalability and accessibility, but it also highlights the need for greater privacy and security. To achieve the latter, it is generally agreed that data stored in cloud servers must be encrypted. Furthermore, the private keys should remain in the hand of data owners. However, this simply stated goal does not have simple solutions. Much of the features and functions that databases and storage systems are accustomed to do not work intuitively with encrypted data as it does with plaintexts. Researchers continue to actively explore solutions for secure storage on private and public clouds, proposing schemes that would enable different functions such as searching, auditing and deduplication.

In terms of search, Boneh [1] proposed one of the earliest work on keyword searching using public key encryption which allows a set of keywords to be searchable without revealing the content of emails. Around the same time, Waters [2] proposed schemes for searching over encrypted audit logs. Although early works focused on single keyword searches, more recent works have focused on conjunctive keyword searches involving multiple keywords [3], [4]. In addition, some considered the problem of ranking of search results [5], [6], [7]. Other researchers provided solutions for searching with keywords that might contain errors [8], [9], termed fuzzy keyword search. The ability to search for phrases were recently investigated [10], [11]. While much work were done on enabling features, security flaws have also been found [12] and solutions proposed [13].

In this paper, we present a conjunctive keyword and phrase search scheme with low storage requirement. The scheme is capable of basic ranking and has the ability to search for non-indexed keywords. We begin by providing the basic conjunctive search algorithm in section III and the basic phrase search algorithm in section IV. An in-depth discussion of the modifications to balance the different security, storage and communication requirements can be found in section V. Finally, we present the results of our scheme applied to a database of text documents in section VI.

II. Model for keyword search over encrypted data

The communication model for a keyword search protocol generally involves up to three parties: The data owner, the cloud server and the user. In a private cloud, the user is simply the data owner. For most of our discussions, we will be considering the public cloud scenario involving three different parties. A typical protocol is illustrated in figure 1 where the user initiates the search request by passing the keywords to the data owner. The data owner then sends a trapdoor to the cloud to initiate a protocol to search over the requested keywords. Finally, the cloud responds to the user with the indexes to the requested documents.

The cloud server usually wields significant computational power compared to the data owner and users. Therefore, it is desirable that the computational and storage cost be asymmetrically placed on the cloud.

A. Security

There are two main security issues that concerns searching algorithms: privacy of the document sets and privacy of

2159-6190/15 $31.00 © 2015 IEEE
DOI 10.1109/CLOUD.2015.74

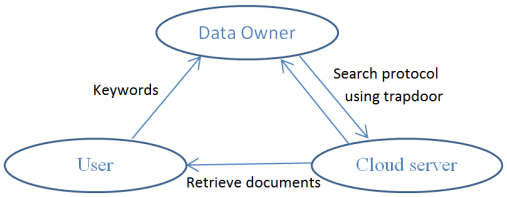

Figure 1. Communication model for keyword search over encrypted data

the searched keywords. As with most existing work, we assume a semi-honest cloud server, which would follow our protocol without deviation but is interested in gathering information from the data that it has access to. Privacy of the document sets implies that the cloud must not learn anything about the documents that are stored on its servers and the user must not learn anything more than the requested documents. Privacy of the searched keywords implies that the cloud must not learn anything about the keywords used to search the documents that are stored on its servers. The latter is a very difficult issue to address if we consider an adversary with some prior knowledge about the documents, such as the language and the content (ex: business, technical specifications), or the pattern of the search requests. Since the user must retrieve the documents at some point, the association of subsets of documents to search patterns is inevitable. Although private information retrieval protocols can be used to lessen its effect, they are costly and can still leak information over time. As with all existing works, we will restrict our security discussions and offer solutions to address search privacy on the scheme prior to document retrieval.

III. Basic conjunctive keyword search protocol

Our starting point is a simple index based keyword search scheme, similar to one used in [11].

For a document collection, $D = \{D_1, D_2, \ldots, D_n\}$, we parse each document, D_i, for a list of keywords, kw_j. An index, I, is then generated mapping keywords to documents such that $I(kw_j) = \{d_a, d_b, \ldots, d_n\}$, where $d_i = 1$ if kw_j is linked to the document. The documents are then encrypted and uploaded to the server. The index is also encrypted prior to being placed on the cloud server:

$$I(E_K(kw_j)) = \{E_K(d_a, d_b, \ldots, d_n)\}. \quad (1)$$

Alternatively, a local dictionary mapping keywords to unique word id's can be used, but must be stored and maintained by the data owner.

To perform a search, the user sends a set of keywords $kw' = \{kw_1, kw_2, \ldots, kw_q\}$ to the data owner. The data owner computes $E_K(kw')$ and sends to the cloud server. The cloud server returns the encrypted index entries to data owner, who then finds the documents matching the requested keywords from the intersection of index entries:

$$I(E_K(kw_1)) \,\&\, I(E_K(kw_2)) \cdots \,\&\, I(E_K(kw_q)), \quad (2)$$

where $\&$ denotes a bitwise *and* operation. Finally, the cloud server sends the matched documents identified by the data owner to the user.

IV. Phrase search based on symmetric encryption

The ability to perform phrase search in encrypted data is only recently explored by researchers. The work by Zittrower [10] was the first to address it in the literature. The proposed solution uses truncation of encrypted keywords to generate false positives in query results to hide the true search terms. As a result, part of the index must be stored client-side and the search is also performed by the client rather than the cloud. The use of false positives to provide security can be unreliable since the number of false positives associated to individual search terms is random. Tang [11] addresses the privacy concerns by proposing a solution with provable security using normalization. The technique uses an index table that allows for verifications of chains of keywords instead of having the word locations stored. However, the index table requires significant storage, which hinders its practicality.

The key difference between conjunctive keyword search and phrase search is that, in addition to containing the requested keywords, they must also appear contiguously in the specified order in the document. In another word, we must have some knowledge on the locations of the keywords. In [10], all keyword locations are stored alongside encrypted keywords in an index. In [11], the relative location of the keywords are stored in a table. Note that a standard single keyword index is also used in both cases. We first observed that the use of a single keyword index alongside the keyword location is enough for determining whether the phrase is present. We then note that we do not need to learn the location of more than a single keyword within the phrase.

To add phrase search capability to the basic scheme, a keyword location index is used. To generate the keyword location index, we compute

$$\{H(D_i|kw_i), E_K(j_1, j_2, \ldots, j_n)\}, \quad (3)$$

where $H()$ is a cryptographically secure hash function and j_x are the locations of kw_i within D_i. Given a user phrase search request $kw' = \{kw_1, kw_2, \ldots, kw_q\}$, the data owner proceeds as in section III to determine documents containing all keywords. It then selects a random keyword in the set and queries the location of the keyword by sending $H(D_i|kw_x)$ to the cloud. Given the locations, the owner returns

$$\{H(E_{K_{D_i,j_s}}(kw_1, kw_2, \ldots, kw_q)), i, j_s\} \quad (4)$$

where i is the index of the matched document and j_s is the identified starting location of the phrase, 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:

$$H(E(w_{j_s}, \ldots, w_{j_s+q})) = H(E_{K_{D_i,j_s}}(kw_1, \ldots, kw_q)). \tag{5}$$

A. Modes of operation

Unlike previous works by [10] and [11], we do not rely purely on indexes to determine matches. Instead, we process the encrypted document themselves on the fly. The advantage is that we require less information to be stored in the index and a lower storage requirement by the cloud server. The ability to process encrypted data, however, poses some challenges. With asymmetric encryption, the computational cost can be prohibitive. With deterministic symmetric encryption, the data must be encrypted using a mode of operation that prevents the same plaintexts from mapping to the same ciphertexts.

Cipher-block chaining (CBC), the most commonly used mode of operation, would require an extra step where the cloud must return the ciphertext, $E(w_{j_s-1})$, directly ahead of the expected starting position of the phrase to the owner in order for the owner to compute $H(E(kw_{j_s}), kw_{j_s+1}, \ldots, kw_{j_s+q}))$. A simple initialization vector can be $H(D_i)$, where D_i is the document id, should a phrase starting at the first position be queried.

Alternatively, we propose the use of counter mode (CTR), which does not require this extra step and also renders the encryption parallelizable, an advantage today where multi-processor computers are ubiquitous. The initialization vector can be stored alongside the document in plain or simply use $H(D_i)$ to provide further saving.

B. Non-keyword search

An interesting property of the proposed scheme is that it has the ability to search encrypted documents for words that are not part of the keyword space. To the best of our knowledge, this is currently the only proposed phrase search algorithm with this ability. In many scenarios, it is impractical to index every word in the document sets. Common words such as 'it' or 'and' are often omitted. Some may choose to go further and index only distinctive words relevant to the document.

As long as the queried phrase contains at least one keyword, the owner can determine the expected starting position of the phrase and proceed with the query. Although possible, querying a phrase containing no keywords would require expensive brute-force matching through the document set.

C. Ranking

The scheme provides the basic ability to rank the returned results during the final step of the phrase matching process: The number of matched phrases per document can be tracked and used to rank the returned results. Though not discussed here, further ranking capability can be added by incorporating TF-IDF as in [14], [15] during the initial conjunctive keyword matching phase using order preserving encryption [16]. This can even be used to provide the best odds of relevant results should an application wishes to limit the number of results for the query by proceeding to the phrase search on only the highest ranked documents in the initial step.

Proximity ranking which was suggested by Zittrower [10], [7] can also be used here since the location of the keywords can be queried.

V. Security

At rest, the cloud server contains the encrypted documents, $E_{K_{D_i}}(D_i)$, the keyword index, $I(E_K(kw_j)) = \{D_a, D_b, \ldots, D_y\}$ and the keyword location indexes, $\{H(D_i|kw_i), j_1, j_2 \ldots j_n\}$. The security of the encrypted documents at rest is equivalent to that of the symmetric encryption $E()$ and cryptographic hash function, $H()$. However, the keyword and location indexes are particularly susceptible to statistical attacks, since certain words are more common than others in every natural language. The location of the keywords may also reveal information to an adversary with partial knowledge [10].

To alleviate the problem, Tang [11] opted to encrypt the single word index and have the keyword chain index normalized by filling in random data so that every entry contains the same number of elements as the largest entry prior to normalization. Encrypting the single word index is sufficient to provide provable security for hiding single word statistics. However, hiding the statistical properties of the keyword chain index used to provide phrase search capability came at a high cost in storage. It is known that most natural languages roughly follows Zipf's law, which states the word frequency is inversely proportional to its rank in the frequency table [17]. By filling in entries up to the maximum occurrence of a word in the entire document set, the resulting index comprises almost entirely of random data. Figure 2 shows the distribution of a particular research paper with approximately 6000 words, which follows roughly the distribution of documents of English language. We'll refer to this example for the remaining of this section to explain the design of our scheme. If all words in the sample document are indexed, the most common word is 'the', with 445 instances, $\approx 7\%$ of all words. It was found that the ten most common words comprises 25% of all words in the document. In fact, almost 1200 of the 1300 distinct words occurred less than 10 times in the document. Using Tang's scheme, the keyword chain table for this document would

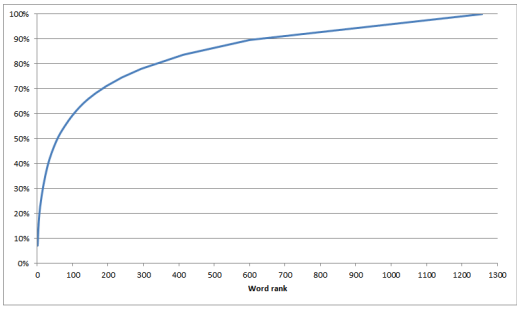

Figure 2. Cumulative distinct word distribution of a sample document

contain 446*1300 entries where over 436*1200 contains random data. Thus, less than 10% is actually used for searching. If the most common English words are excluded, we would have 1066 distinct words indexed and the 20 most common words comprising 25% of all words indexed. The most common word occuring 84 times and over 1000 words occuring less than 10 times, resulting in less than 26% of the data useful for search.

During query, one must also account for the statistical property of search terms, that is users may be more likely to perform a search for a subset of keywords. Then, it may be possible for the cloud server to establish a known plaintext scenario, where kw_j and $E_K(kw_j)$ are discovered. Since the indexes, d_j, are encrypted, the link between document and keywords are not available from the index only.

A. Reducing storage cost in providing security of the location index

The algorithm provided in section IV, although efficient, is vulnerable to statistical analysis. Namely, the number of entries for each word in a document is not hidden. A modification can be made to provide similar privacy protection as in Tang's scheme [11]. The goal of normalization is to hide the fact that certain keywords appear more frequently than others. Since the number of entries for each word cannot be reduced without losing potential matches, the simplest approach would be to insert random data until every word is associated with the same number of entries, as suggested by Tang. Furthermore, to hide the statistics of the most frequent word of each document in the document set, the normalization can be applied over the entire set rather than per document. This results in every query returning the same number of entries independent of the keyword or document, but at a high storage cost.

As noted in section V, keyword frequencies are distributed such that most common words dominate the entries. Although not as pronounced when the most common words in English are excluded, the trend remains. In the example provided, the two most common words were 'decoding' and 'algorithm', the topic of the research paper, appearing 84 and 64 times respectively. By the 21st most common word, the count has reduced to 18. Therefore, the high cost in Tang's scheme is due to having to adjust the majority of the keywords to accommodate these very few common keywords. Perhaps it would be preferable to do the reverse. Instead of increasing all entries to match the most common keyword, we could 'reduce' the number of entries taken up by the most common keywords.

Rather than normalizing according to the most common keyword in the document set, we could normalize to the keyword at, say, the $p = 95\%$ percentile when ordered from the least to the most common. For 95% of the keywords, it suffices to increase the number of entries to match the target keyword. For the most common $1-p = 5\%$, each keyword's entries must be split into multiple equivalent keywords, each comprising of the target number of entries. In the example, $p = 95\%$ corresponds to a keyword which appeared 10 times in the text. Keywords that appear between 11 and 20 times will be split into 2 equivalent keywords, etc. This results in $1 * 31 + 2 * 7 + 3 * 4 + 4 * 4 + 5 * 1 + 6 * 2 + 8 * 1 = 98$ extra keywords, each with 10 entries, for a total of $1164 * 10 = 11640$ entries in addition to the 1164 indexed keywords. If normalization to maximum keyword had been used, the resulting table would contain $1066 * 84 = 89544$ entries instead, a 87% reduction in number of entries in the proposed approach. In fact, the reduction in storage cost is greater still since an entry in a keyword chain index requires two parts containing the hashed previous word's location and the current word's location while our scheme requires only the latter.

Note that a split keyword requires more storage for its different variations and for padding the number of entries to a multiple of the target value. It is also necessary to store the number of times a keyword's been split. As p decreases, the amount of storage we save increases, but at a slower rate, potentially reaching a point where the momentum reverses and storage cost instead increases. Depending on the keyword frequency distributions, the optimal value for p can be any value between 0 and 1. At the limit case of normalizing according to the least common keyword, reasonably assumed to have only a single entry, the scheme reduces to indexing every word in the document, resulting in 3224 entries with 3224 keywords in the example. Assuming that keywords require same storage as a location entry, the optimal value for p was heuristically found to be 70%, where 2010 keywords were used with each containing only 2 entries. A table listing the number of times 149 keywords is split must also be stored by the client. Note that the client side storage cost is comparatively low due to the low number of common keywords in typical documents. Section VI contains a more in-depth discussion and experiment to find the optimal value for p and an optimal split value.

Another advantage of this approach is the ease in adding and removing documents from the set. A new document

can be parsed as usual and each keyword index is split and padded to fit the norm of the set. Conversely in [11], a problem could arise where a new document's most common term appears more frequently than the most common term in the document set, and could not be added without regenerating the index. Removing documents is also straight forward, with each document corresponding to a bit in the single keyword index and each document having its own keyword location index. The task becomes more difficult using Tang's approach since the keyword chain index is applied to the entire document set. Note that our scheme does leak information on the statistics on the number of distinct keywords for the documents. However, since its value is typically a function on the document size [18], which is available to the cloud server, we do not believe it to be a significant source of security leak. Therefore, we did not normalize the number of distinct keywords between different document indexes.

B. Reducing communication cost in providing security of the location index

There's also a communication cost to querying more keywords. However, since the cloud server does not need to return a significant number of unused entries, the communication cost is also reduced. For the sample research paper, a query for 'polynomial code' requires returning only 37 entries using the proposed scheme whereas 168 entries would have to be returned if maximal normalization were used. On this note, one may notice that the number of entries queried are susceptible to statistical analysis. If a query for a single keyword returns a very small number of entries, one can reasonably assume that the queried keyword(s) is not a common word. However, if a query for multiple keywords returns a larger number of entries, it does not imply a common keyword was queried since multiple keywords can be assigned to a common term or many less common terms. There is no obvious way for the cloud server to decide whether two encrypted keywords are equivalent. Similar to previous case, maximal normalization would provide guaranteed protection. Assume the most common keyword in the document set is associated with x equivalent keywords, then a query for n keywords should proceed as follows:

a) Select random $nx - n$ keywords
b) Perform the query for the combined nx keywords

In this way, every keyword queried returns the same number of entries. Based on similar reasoning as for storage, we argue that we would achieve only marginally less protection for common keywords if we instead set a minimum of $q \leq x$ keywords queried instead of nx. A query for $q = x$ keywords can then be associated with the most common keyword or a combination of many less common keywords. From the example, less than 2% of the keywords appear more than 18 times while the most common term appears 84 times. Choosing $q = 18$ would then only reveal that a query of 18 keywords does not include 2% of the keywords in the document. Therefore, choosing $q < x$ could provide significant savings at the cost of minimal security loss.

VI. Analysis

To provide a better understanding of the scheme on large document sets, we retrieved 1500 documents made available by Project Gutenberg [19] and compared our results against other phrase search schemes. The Natural Language Toolkit [20] was used to determine the statistical properties of the corpus and the performance for the various schemes.

A. Effect of common passages in a corpus

The retrieved documents often include headers and footers outlining the copyright, contact and source information. The inclusion of such texts that are repeated throughout the document set can skew the statistical property of a corpus. Other examples of such documents include forms, contracts and technical documentations. The possible inclusion of repeated texts highlights the need for hiding the word frequency distribution of documents. To better understand the skewing effect, we attempted to remove the header and footer from the documents. However, due to inconsistent placement and wording, we were only able to remove most, but not all of them.

While examining the frequency distribution of words in each document, we found that the most frequent word in each document was particularly affected by the skewing effect. The following are the most common words in the list when headers and footers are included: said, would, *ebook*, upon, thou, king, state, *etext*, love and like. When they are removed, the list becomes: said, would, upon, thou, king, love, state, like, little and could. Note the terms, *ebook* and *etext*, are frequently used words in the header and footer rather than the content. Further examinations also revealed that, while the corpus averaged 4000 distinct words per document, the documents whose most common words were ebook or etext contain an average of only 1600 distinct words. Therefore, the skewing effect affects mainly shorter documents where the header and footer represent a significant portion of the words.

Should the basic scheme in section III be used without any protection against statistical analysis, an examination of the index tables could reveal significant portion of the content of the shorter documents by matching the frequency distribution of words in the header and footer against the encrypted keywords, $E_K(kw_j)$, in the index table. We illustrate such an attack in the following scenario: A company is storing application forms on a cloud service provider. An employee of the cloud service provider legitimately uses the company's service to submit a form. In addition to having knowledge of the form's content, the employee also has access to the corresponding encrypted file along with the encrypted index tables. Any texts that are not entered by

a user must be in every encrypted form. If the amount of texts entered by the user is much less than the texts already on the form, the employee could have reasonable chance of success at determining the keywords, kw_j, corresponding to, $E_K(kw_j)$, by comparing the number of entries for the keywords with the frequency distribution of words over the form. Any discrepancies must then be user-entered texts.

B. Effect of indexing common words in a corpus

Common words such as "the", "on" and "at", also called stop words, are often filtered out before indexing since they are functional terms of little relevance to the document's content. Indexing common words can also be expensive in both storage and computation. However, filtering them out could limit the ability to search for phrases that include such terms, as is the case in [10], [11]. Recall that our scheme can search for non-indexed keywords. To evaluate the effect of indexing common words, We processed the corpus with and without stop words.

When common words are included, the most frequent word in 96% of the documents becomes "the". In terms of security, if protection against statistical analysis is not included, this can easily be exploited as described in previous section. In terms of storage, we observed that the highest number of instance of "the" in a document was 129070. When stop words are removed, the number of instance of the most frequent word becomes only 10757, which is 92% less frequent than "the". This illustrates that not indexing common words can lead to significant reduction in storage cost. To correctly search for phrases that include stop words, the approaches by [10], [11] would require indexing common words, which translates to significantly more storage than our scheme.

C. Finding an optimal p value relative to storage cost

We examined the optimization of the scheme in terms of storage. Since symmetric encryption is used, storage cost of the encrypted documents is optimal. The storage cost of our keyword-to-document index is similar to existing schemes. Since there's only one such index for a corpus, its cost is relatively small compared to the location index which is assigned to each document. Therefore, we focus our evaluation on the location index. To this end, we first define the storage cost to be the total number of entries the table maintains. The storage cost of the scheme is dependent on the amount of times a keyword is split. We initially defined a p value which determines the percentage of the keywords in a document that are left unsplit. Figure 3 shows the histogram of optimal p value for the sample document set. For 90% of the documents, the optimal p value was observed to be between 60% and 70% percentile. The histogram for the corresponding number of entries assigned to each keyword is shown in figure 4. The majority of the optimal split values are between 2 and 4. Although the p value was found to be more consistent throughout the corpus, it might be more practical to have a fixed split value to reduce the preprocessing cost. It may also be more secure. Since the optimal split value is dependent on the keyword distribution of the document, Indexes with different split values could leak information on the document content.

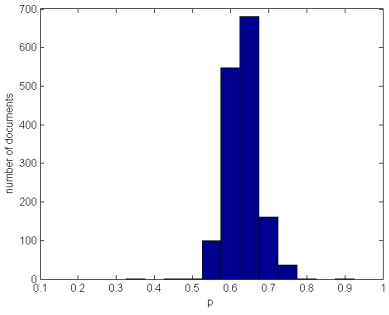

Figure 3. Optimal p value across 1500 sample documents

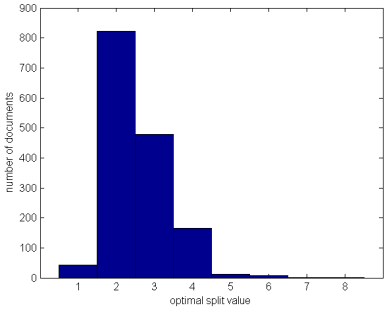

Figure 4. Optimal split value across 1500 sample documents

D. Comparison with other schemes

We compare our phrase search scheme to Zittrower's [10] and Tang's [11]. Since symmetric encryption is used, the encrypted documents require roughly the same amount of storage as when in unencrypted form. This is true for all three schemes in consideration.

Using Zittrower's approach, the user must store a dictionary mapping every distinct word in each document to an index value used to distinguish between different keywords since different keywords can map to the same entries in the index table. Suppose there are x distinct keywords in the corpus and suppose optimal representation for the index value was used, this represents $x(log_2(x)+b)$ bits of storage, where 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. Each encrypted keyword was truncated to 12 bits. Suppose that a location value requires y bits, that there's an average

of x' distinct keywords and that each keyword appears q times on average per document. Then, the location table requires $x'(12+qy)$ bits and the keyword-to-document table requires $x(plog_2(n)+12)$, where p is the average number of documents associated with a keyword in the corpus and n is the total number of documents in the corpus. The keyword-to-document table here requires the least amount of storage among the three schemes since normalization was not used. Instead, the scheme relies on false positives to defend against statistical analysis. The author stated an average of 300 collisions among the encrypted keywords when 12 bits are retained. This means a query for a single keyword would on average also return results belonging to 300 other keywords that must then be processed by the client. Although it is possible to reduce the communication cost by increasing the number of bits retained, this leads to lower collision rate and susceptibility to statistical analysis. Although it does not lead to a higher storage cost, the technique requires a higher communication and computational cost, especially on client side. Since the amount of false positives generated is random, the technique also has the risk of generating very low number of false positives for certain keywords, leading to a variability in security throughout the keyword set.

In Tang's scheme, the user similarly stores a dictionary mapping keywords to index values, requiring $x(log_2(x)+b)$ bits of storage. The server also stores two tables: The keyword-to-document index, requiring $x(log_2(x)+n)$ bits of storage, and a location index table requiring $x'(h+d(u+y))$ bits of storage, where h is the number of bits to store a hashed keyword, u is the number of bits to store a hashed 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 location index table is several orders of magnitude greater, due to the normalization to the value of d. The goal of the normalization was to achieve security against statistical attacks, at a high storage cost. Unlike in Zittrower's scheme, the technique allows the cloud to perform the majority of the computations. In terms of communication, the scheme includes returning irrelevant results since normalization requires insertion of random data into the index tables, although much less than in Zittrower's scheme.

In our scheme, the user must also keep track of the keywords that are split in addition to the dictionary mapping keywords to index values. Our experiments showed that splitting keyword entries into pairs were ideal for reducing the size of the location index tables on server. Splitting into pairs, on average, increases the number of keywords by 150% in the location table, where approximately 27% of the keywords were to be split. Suppose that there is an average of x' distinct keywords per document and that k is the average number of instance of most frequent word per document, then the split keyword table would require $0.27x'(log_2(x') + log_2(k/2))$ bits of storage. Note that this client side table can be reduced in storage by splitting into triplets or quartets instead, which would respectively reduce the number of keywords to be split to 18% and 14%, although it would also have the effect of increasing the server side storage cost. It is also possible to encrypt the split table and store on the cloud similar to the index tables, eliminating the need for client side storage aside from the keyword dictionary. However, it must be retrieved in entirety and decrypted during search. Retrieving only split values of the relevant keywords would leak information on the number of common words among the search terms since only common terms are split. On the cloud server, our scheme requires a keyword-to-document index table which uses $x(log_2(x)+n)$ bits of storage and a location index table that requires on average $2.5x'(h + 2y)$ bits, where h is the number of bits used to represent the hashed keyword. Although our scheme requires a higher client-side storage than both Tang's and Zittrower's schemes, the size of the split table is significantly smaller than the normalization cost in Tang's scheme and far less irrelevant data are used in normalizing the tables as explained in section V-A.

Table II summarize the results of the schemes on the sample Gutenberg document set with the experimental values for the various parameters outlined in table I. For the hash values of keywords and locations, we assumed that each would require 16 bits in all cases. Since there are approximately 1 million words in the English language today, we would need no more than 20 bits to guarantee no collision. We believe 16 bits is reasonable since the number of words used in most scenarios is far less than the limit. As shown, the proposed approach can achieve significant savings in cloud storage at a modest increase in user storage.

VII. Conclusion

We discussed various design issues on security and efficiency related to phrase search algorithms on encrypted data and presented a conjunctive keyword and phrase search scheme that achieve a low storage and communication cost relative to the current state of the art phrase search algorithms. Based on our experiments, the proposed solution achieves 3 times less storage than the current leading solution in storage requirements while also enjoys the high level of document and keyword security demonstrated in [11] without the prohibitive storage cost. The scheme provides basic ranking capability and is also currently the only phrase search scheme that can work with non-indexed words.

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.

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 distinct keywords per document, x'	3959.6
Average number of times each keyword appears per document, q	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 OF PHRASE SEARCH SCHEMES FOR A SAMPLE OF 1500 DOCUMENTS

	Zittrower[10]		Tang[11]		Our scheme	
	user	cloud	user	cloud	user	cloud
Storage	1.2MB	392.5MB	1.2MB	242.8GB	5MB	139.3MB

[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.

[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] Zhangjie Fu, Xingming Sun, N. Linge, and Lu 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] Charles L. A. Clarke, Gordon V. Cormack, and Elizabeth 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] 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.

[9] 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.

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

[11] 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.

[12] Hyun Sook Rhee, Ik Rae Jeong, Jin Wook Byun, and Dong Hoon 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.

[13] 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.

[14] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou, "Secure ranked keyword search over encrypted cloud data," in *International Conference on Distributed Computing Systems*, 2010, pp. 253–262.

[15] J. Zhang, B. Deng, and X. Li, *Additive Order Preserving Encryption Based Encrypted Documents Ranking in Secure Cloud Storage*, pp. 58–65, Springer Berlin Heidelberg, 2012.

[16] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu, "Order preserving encryption for numeric data," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004, pp. 563–574.

[17] David M. W. Powers, "Applications and explanations of zipf's law," in *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, 1998, pp. 151–160.

[18] L. Egghe, "Untangling herdan's law and heaps' law: Mathematical and informetric arguments," *Journal of the American Society for Information Science and Technology*, vol. 58, pp. 702–709, 2007.

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

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