Encrypted Phrase Searching in the Cloud

Steven Zittrower and Cliff C. Zou
Dept. of Electrical Engineering & Computer Science, University of Central Florida, Orlando, United States
steven.zittrower@cs.ucf.edu, czou@cs.ucf.edu

***Abstract*—As cloud computing is increasing in popularity, it is difficult to both maintain privacy in datasets while still providing adequate retrieval and searching procedures. This paper introduces a novel approach in the field of encrypted searching that allows both encrypted phrase searches and proximity ranked multi-keyword searches to encrypted datasets on untrusted cloud. By storing encrypted keyword-location data along with specially truncated encrypted keyword indexes in a relational database, we are able to allow for a full range of search features in our encrypted searches, something that has never been accomplished before. Furthermore, our approach permits the encrypted corpus and index to both be stored on cloud data servers. We modify currently available open-source search engine software to complete a prototype and provide results from experiments on a large scale real-world dataset that has more than half a million documents.**

I. Introduction

Encryption is a method that secures information by making it illegible or indistinguishable from random noise to anyone that does not have some privileged information, a key. The practice of using cryptography to encrypt sensitive information has been around for millennia. For thousands of years a major tenet was that the encrypted information was unusable until decrypted. This served well until recent, when the vast number of documents needing to be encrypted has made decrypting individual documents to find query results infeasible in practice. Searchable encryption was invented to solve the problem of how to find keywords in documents that are encrypted without decrypting the entire corpus set.

Searchable encryption is not a new concept but all current methods have failed in various aspects that keep them from becoming common or mainstream. Most proposed methods utilize advanced mathematical structures such as Bloom filters or trap-doors but they typically only allow for Boolean searches and do not support phrase searching [1] [2] [3]. Subword matching, exact-matches, regular expressions, natural language searches, frequency ranking, and proximity-based queries are all forms of searching that modern search engines employ and users expect to have. For example, being able to search for 'heart attack' and distinguish between results related to myocardial infarction (heart attack) as opposed to 'an anxiety attack that caused heart palpitation' is important for the adoption of any searchable encryption scheme. Current methods are incapable of performing this kind of phrase searching. Even ranked word proximity searches, a search that ranks results based on how close the query keywords are together, has not been fully implemented by previous research. The closest previous methods stop at ranking documents only by the number of times a keyword appears in the documents and whether the keyword is within pre-defined locations [4].

A recent string of widely publicized data breaches in the cloud, such as an attack on the Gmail accounts of U.S. government officials [5] and a massive attack on Sony where thousands of credit card numbers and millions of customer's personal information were lost [6], have shown that third-party cloud providers cannot be fully trusted to safeguard data. These attacks have further illustrated the need for securely encrypting private information.

By making use of a trusted client-side server to encrypt and decrypt words and metadata we can store our files and search index offsite on untrusted clouds while ensuring the integrity of the encrypted data and fulfilling potential government or industry regulations on sensitive data.

We consider the scenario where an organization outsources its internal data to a public cloud, or generally speaking, we consider the case where the data owner has the authority and ability to set up a trusted proxy for access of its encrypted data in cloud. All of the organization's employees are considered as 'clients' for the encrypted data search. It would be easy and practical for the organization to set up a trusted proxy between the cloud and its employees. All encryption and decryption of the encrypted data stored on cloud is done by the trusted proxy; that is, clients have no knowledge of the security keys. Thus, if an employee's computer is compromised or the employee is an inside attacker, once permissions of this employee have been revoked, the encrypted data is still safe on the cloud.

This paper presents a novel approach to allow for phrase searching and query proximity ranking for search queries on encrypted data in the cloud. Neither the source documents, nor the search index database needs to be hosted on local or trusted servers. Both of them will be encrypted and hosted in remote public cloud servers. Further contributions of this paper include: a complete prototype of our proposed search methods using a realistic large-scale dataset of over 500K documents in which searches can be completed within seconds and a comprehensive experimental evaluation of the overhead, speed, bandwidth, and security of our method.

The rest of the paper is organized as follows. Section II discusses related works and previous research in encrypted searching. In Section III we present our method for encrypted phrase and proximity ranked searching and discuss the implementation of these methods in Section IV. We analyze our results and evaluate our design in Section V. We conclude in Section VI.

II. Related Works and Background

In the past, most keyword searching encryption schemes focused on a searchable index of words which remain hidden to the server until a one-way trapdoor function is given [7]. These indexes have more recently been stored as a Bloom filter to decrease the size of the index and increase the security by reducing the chance of active server attacks by allowing for false positives [2] [3] [8]. In [8], Bellovin and Cheswick published a method for storing an index using Bloom filters and its corresponding cipher functions such that neither the querier nor the receiver knew of each other's search or collection. Bellovin went on to modify this paper in [3] to support database searching functions. Most related works in this field focus on single-keyword searches or Boolean searches. However, in the past few years, papers on multi-keyword ranked searches [9] and very limited proximity ranking [1] have been published.

In [2], Aviv, et al., developed a method to securely search files on a remote storage based server. They encrypt and hash the words into a Bloom filters thus keeping the index secure. However, their method provides for no way of including phrases or proximity ranked results as their index does not store any keyword location data.

In [9] and [10] this method was further improved by creating a design that supports multi-keyword searches and ranked searching. Although this is a leap-ahead of previous work, as it provides ranked results and multiple keyword searching, it still fails to include any ranking based on word location proximity or phrase searching capabilities.

Encrypted database research, specially that in the area of the publisher/subscriber model, has produced interesting results that is corollary to our research. Raiciu, et al. [11] and Srivatsa, et al. [12] both developed methods of maintaining confidentially in a Content-Based Publish/Subscribe. While our model technically only has one subscriber (the trusted client-side server), further research in this area could potentially allow for direct cloud-to-client communication.

Another area of research that is similar to ours is public-key searchable encryption. Boneh et al. [13] published a method to allow a public-key encrypted document's owner to provide a gateway server a specialized trapdoor to test whether a word is contained in the owner's encrypted documents. Many of extensions have been given to allow multiple keywords [14] and conjunctive searching with keyword subsets [15]. While these are similar to our end goal they are very limited in scope as the documents must be encrypted with a single user's private key and keywords to search for must be known in advanced. Also, while conjunctive and disjunctive searching is possible, proximity ranking is not.

Current research focuses on two fronts: adding features to previous designs (such as keyword ranking, proximity ranking, predicate and disjunctive searches and Boolean search operators) [9] [16] [17] and creating more computationally efficient encrypted searching designs [18].

The closest design that does support proximity ranking was introduced by Artzi, et al, in [1]. It divides each document into 64 separate sections and stores the hashes of this metadata along with each keyword separately in a Bloom filter. This allows the search engine to tell whether a word is within a generous range of another. While this does support limited proximity ranking it still lacks support of phrase searching.

Our model is unique and novel as it implements phrase and proximity based ranking of search results. It abandons using a Bloom filter to store document information in favor a relational database. Our method trades the small storage size and the secure nature of Bloom filters for a more flexible, but larger, data structure. We then take extreme caution and diligence to prevent information leakage related to active and passive attacks on the database, search query, and search results.

Where previous models were limited in their search functionality and, therefore, unlikely to be used in commercial settings our model allows for a full complement of modern search features including but not limited to stemming (reducing words to their root), lexicographic parsing, Boolean searching, phrase searching, and frequency and proximity ranking of the results. Currently, no other research offers a full set of these features.

Some prior research on encrypted searching do not have a trusted client-side server in their architectures [4] [9]. We believe it is practical and reasonable to assume that the data owners can set up their own trusted servers locally, which is not difficult for a data owner and does not introduce much overhead cost. In addition, the trusted local server can greatly simplify the networking architecture and protocol design and improve data security against insider attacks. Most importantly, we believe the simplicity of the proposed architecture will make it easier to be understood and accepted by company management teams to implement data outsourcing to the cloud.

III. Proposed Encrypted Phrase Searching

"Proximity ranked searching" implicitly ranks documents by a function, f, that is directly proportional to the distance the keywords in the search phrase appear from each other. We adapted [19]'s keyword ranking algorithm and modified it to allow for more than three keywords. In addition, we implement search querying techniques presented in previous research such as Boolean searching and multi-keyword ranking.

To allow for proximity ranking, the location information of keywords must be preserved in the encrypted index created from the data corpus. This is a challenge that previous research relying on Bloom filters cannot easily overcome. Instead of using a highly compressible index, such as a Bloom filter, we make use of a relational database to store the three valuable components to each document: document reference, keywords, and keyword locations.

A. Overview

Our architecture makes use of an encrypted index which is generated prior to searching. A trusted client-side server generates the encrypted index and transfers this index to an untrusted cloud server. Further details on this index and how to create it are discussed in Section III-B.

ID	Content	Rank
3	…it was a *disease* of the *heart* that induced …	2
5	…*heart disease* is the leading cause of death in the …	1
10	…*diseased* ideas led to him becoming *disheartened* …	3
13	…his small *heart* grew three sizes that day …	N/A

Table I: Four example documents sorted by ID and their suggested ranking for search query 'heart disease'

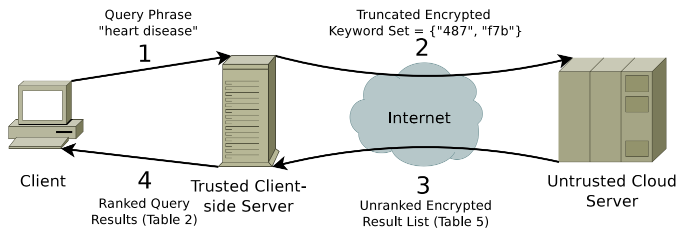

Fig. 1: Flowchart of the proposed encrypted phrase searching procedure. For a company that outsources its dataset to an untrusted public cloud, a 'trusted client-side server' is set up internally (several identical trusted client-side servers can be set up for robustness purposes). 'Client' represents any company employee or user.

Fig. 1 outlines the general process of our proposed encrypted phrase searching. First, the client sends a plaintext search query to a trusted client-side server (step 1). The client-side server encrypts all keywords in the search query individually using symmetric-key encryption; it then truncates the encrypted keywords to a set number of bits to improve security by allowing for collisions, and queries the untrusted cloud server for the documents containing the set of truncated encrypted keywords (the order of these encrypted keywords will be randomized) (step 2). The cloud server does a database query of its encrypted index and returns to the client-side server encrypted data that corresponds to document paths, truncated encrypted keyword index offset, and encrypted keyword locations (step 3). The client-side server decrypts this data first. From the newly decrypted keyword index offset it can then determine which returned results are actually for the keywords searched and which are simply collisions. It discards those collisions and filters and/or ranks the pertinent returned documents based on relevant keyword locations and frequency. Finally, it sends this ranked listing to the original client (step 4). At this point, the user can peruse the results for the desired documents and, if desired, perform a request to the untrusted cloud server to retrieve the desired encrypted documents.

B. Indexing

In our model, prior to searching for a document δ, an encrypted index of the corpus must be generated by the trusted client-side server. The index is then encrypted and sent to the untrusted cloud server. Searching takes place by running SQL queries over the encrypted index.

For illustration purposes, a small representation of such an index based on Table I is shown in its unencrypted form in Table II and the corresponding partially encrypted version is shown in Table III. Each row in the encrypted index table corresponds to one document $\delta \in corpus$. Each row contains two columns: an arbitrarily assigned unique document id (ID) and a specialized data structure that contains truncated symmetric-key encrypted keywords associated with encrypted versions of the keyword's location in δ (Word Vectors). In addition, this data string contains an offset that is used to map the truncated encrypted keyword with its full version (stored on the trust client-side server). Table III depicts the offset and the keyword locations unencrypted to better show the structure of the string. In actuality, these characters are concatenated and encrypted together using a block cipher, thus making it unfeasible to determine the offset or keyword locations without the key κ. Without loss of generality, we assume that the cryptographic keys used for both encryptions are the same key κ and that κ can be used for decryption as well. Only the trusted client-side server has access to the value of κ.

Once the encrypted index is transferred to the untrusted cloud server, an inverted index, Table IV(a), based on the encrypted index, is generated by the cloud server to facilitate the searching speed of the index.

C. Keyword Truncation

A main attack point on the proposed scheme thus far is that it is highly susceptible to statistical frequency analysis attacks. If each keyword were encrypted individually using a deterministic encryption method, a nosey cloud provider could compare the encrypted index with a language probability table to estimate which words map to which encrypted words. To combat this problem we truncate the encrypted words to a predefined number of bits β. Therefore, numerous collisions are created since the entire encrypted keyword space size has been reduced to 2^{β}, as seen in Fig. 6.

This method can also thwart a separate analysis attack based on multiple keyword searches. Since keywords are likely to be related to each other it would be possible for a malicious cloud provider or user to create associations between multiple encrypted keywords. If even one encrypted keyword is decrypted, such a table could reveal a large amount of information about the data. To illustrate this point, take the

ID	Word Vectors				
3	diseas:**3**	heart:**6**	induc:**8**		
5	heart:**1**	diseas:**2**	led:**5**	caus:**6**	death:**8**
10	his:**1,6**	diseas:**2**	idea:**3**	led:**4**	employe:**7**
	becom:**8**	heart:**9**			
13	his:**1**	small:**2**	heart:**3**	grew:**4**	three:**5**
	size:**6**	day:**8**			

Table II: Unencrypted representation of the database that the cloud server stores. Keywords are to the left of the colon and the location in the documents are in bold. Multiple locations are delimitated by commas.

ID	Word Vectors				
3	f7b:0—**3**	487:0—**6**	477:0—**8**		
5	487:0—**1**	f7b:0—**2**	55d:0—**5**	d37:0—**6**	ff3:0—**8**
10	110:0—**1,6**	f7b:0—**2**	aef:0—**3**	55d:0—**4**	7e9:0—**7**
	498:0—**8**	487:0—**9**			
13	110:0—**1**	99f:0—**2**	487:0—**3**	2f3:0—**4**	498:1—**5**
	667:0—**6**	eef:0—**8**			

Table III: Partially encrypted version (keyword index and location left in plaintext for illustration purposes) of Table II that is stored on the cloud. Keywords and locations have been encrypted with a symmetric-key cipher. The encrypted keywords have been truncated to 12-bits.

Enc. Word	ID
487	3,5,10,13
498	10,13
...	...
f7b	3,5,10
ff3	5

a: Encrypted

Word	ID
heart	3,5,10,13
become/three	10,13
...	...
disease	3,5,10
death	5

b: Unencrypted

Table IV: Inverted index generated from Table III that maps keywords to documents. (a) is stored on the cloud. (b) is unencrypted and is not stored anywhere but is for illustration purposes only.

following example where, for simplicity, encrypted words are only 4 alphanumerics long. Without this feature, if the search terms 'United Nations', 'United States', 'United Airlines', and 'United Healthcare' with unique encrypted keywords returned of 'ABCE 1A3B', 'ABCE 6C8D', 'ABCE 2E4F', and 'ABCE 5A7B' respectively, had previously been searched for, an attack could store this information. It could be inferred that the encrypted value 'ABCE' is likely a common preceding word or adjective, thus an attacker has gained information about a distinct encrypted keyword. In addition, with enough analysis of this type if it is ever determined that 'ABCE' corresponds to 'united', each other encrypted word in the searches are now known to likely be a term related to 'united'. Having a random, many-to-one, mapping of keywords to encrypted keywords prevents attacks of this nature from reducing the security of the index.

The trusted client-side server creates a unique keyword truncation index value for each encrypted keyword. A table that is stored on the client-side server maps the truncated index value with the fully encrypted keyword. Each index number is stored along with the keyword locations in the encrypted index. This string is encrypted using AES and a randomized salt. As many multiple keywords now map to the same bits it makes any statistical frequency analysis attack far less useful.

The security gained from the truncated encrypted keyword collisions does come at a cost though. Since it is impossible, prior to decryption, to determine which keywords in the collision set were actually being searched for, they must all be returned, decrypted, and then filtered. Therefore, as shown in Fig. 3 and Fig. 6, the more collisions there are the more bandwidth and computation power is required per search.

ID	Encrypted Result	
3	f7b:**488Burh1fH**	487:**F1vFFNWp=**
5	487:**AjDL7i1Bo==**	f7b:**1tsaFlvlBhY**
10	f7b:**XnSh0NB+u**	487:**JnhVD5N7a**

Table V: Encrypted unranked results generated by the cloud server and returned to the trusted client-side server from the search phrase 'heart disease'. The keyword truncation index and the location data are shown encrypted (and in Base64). ID 13 is not returned because it does not contain both of the search terms.

D. Searching

When the search begins, the client sends the query phrase with multiple keywords, $k_1, \dots, k_n$, to a client-side server, which concatenates the keywords to a list, K (step 1 in Fig. 1). The client-side server then encrypts each $k \in K$ using κ in which the order of keywords is randomized. Each keyword in this list is truncated to β bits to create the encrypted keyword list, K'. In our reference example shown in Fig. 1, the client searches for '*heart disease*'. Therefore, with β=12 bits, this phrase is transformed into '*487 f7b*'. The client-side server then transfers this encrypted query, K', to the untrusted cloud server (step 2). In Section IV-C we discuss other enhancements that can be implemented during this phase, such stemming (reducing words to their root) and the removal of commonly found words.

The untrusted cloud server parses K' into individual encrypted keywords k' and, using the inverted index (Table IV), determines the documents, δ, that contain a k'. It then references Table III to find the encrypted location(s), l'_i, for each k'_i in δ. In our example, only documents 3, 5, and 10 contain both of the encrypted keywords so these IDs, the keyword locations, and the truncated keyword index, as shown in Table V, are sent back to the trusted client-side server (step 3).

The client-side server parses the results that the cloud server returns, which include the document's IDs, paths, and associated encrypted keywords, index, and encrypted locations l'. Each l'_i is decrypted to the truncation index τ and l_i. A proximity ranking function R, hosted by the client-side server, is utilized to meaningfully rank the results, as shown in Table I. This proposed ranking algorithm is presented next.

IV. Implementation

A. OpenFTS

To build a working implementation of our encrypted phrase ranking search we modified and rewrote the free and open source application OpenFTS written and designed by Bartunov, et al. [20]. OpenFTS is a full-text search engine fronted written in Perl for PostgreSQL's tsearch2. Since the functionality of tsearch2 is inherently made for natural languages and not for encrypted data and custom data structures, it has strict limits on the size of the database rows and type of data allowed in it. MySQL has looser limits on these and therefore, when rewriting OpenFTS, we modified it to use MySQL.

OpenFTS's Perl frontend allows custom configuration of the indexer, parser, stemmer, dictionaries, search functions and other cosmetic methods that are not of integral importance to the novelty or functionality of our search method but instead show that our method can be extended to include all of these modern searching devices.

The indexer was completely rewritten to encrypt the corpus of documents prior to sending them to the cloud server. The search function wrapper was modified to encrypt the query and decrypt the results so they could be ranked and displayed to the end user.

B. MySQL Full-Text Search

Full-Text Search is a natural language text search engine integrated into MySQL natively since version 3.23. It provides the functions and storage of the encrypted documents, location values, and document names.

We chose to use this implementation for its lack of restrictions on the Full-Text data type field and its support of inverted indexes. Whereas, a traditional index maps keys to values (or documents to words in our case), an inverted index reverses this and maps words to the documents that contain them (Table IV). Inverted indexes are a necessity for our search engine to be scalable and efficient when indexing a large corpus of documents. The MySQL source code was modified to restrict indexing only to the truncated encrypted keywords.

C. Implementation Outline

1) Indexing: Indexing of the corpus takes place prior to keyword searches and is performed by the client-side server. Continuous indexing is possible because each document is self-contained in its own encrypted form, as shown in Table III. However, recreating the inverted index is a time-consuming task, therefore, it is assumed that the index is updated at an order of magnitude less frequent than keyword searching.

Perl scripts parse each unencrypted plaintext document into its root stems. For example, *diseases* and *diseased* would both become the keyword *disease*. At this point, very common words, such as the conjunctions *and* and *or*, as well as other words that appear in most documents but provide little insight or distinguishing value are removed (these common words are called *stop-words*). We encrypt each keyword using a symmetric-key cipher. The location of each keyword in the document is also encrypted. In our implementation, we used AES for its security (any kind of symmetric encryption algorithm can be used in practice). The encrypted keyword is then truncated to a specified number of bits, β. The client then searches the truncated index array for the index number corresponding to this encrypted keyword. If the full encrypted keyword is not already contained in the index array it is appended to the end and the truncated index becomes one greater than the previous maximum truncated index for the corresponding bits. The index and location data are salted and encrypted via AES and appended to the encrypted keyword. It is then transferred to the cloud to be stored in a custom-made MySQL structure. Finally, once the encrypted index is transferred to the untrusted cloud server an inverted index of the table is created to allow queries to be a function of the number of unique encrypted keywords.

2) Searching: Our implementation of searching closely follows that described in Section III-D. Just as in indexing we modified the OpenFTS code to parse the keyword string for stop-words and stem each keyword to its root. We then encrypt and truncate each keyword using the same AES algorithm, salt, and β value that were used in indexing. A SQL query is formed and transferred to the cloud to run against its database.

The query is first compared against the inverted index to find which documents (and their corresponding rows in the index) contain the keyword set. Then these rows are analyzed and the encrypted truncated keyword index and keyword locations are returned to the client-side server.

Our modified OpenFTS code then decrypts the keyword index and locations using the symmetric-key that they were encrypted with. It discards any results with truncated keyword indexes that do not correspond to the searched keyword and then calls the ranking function. All document ranks are normalized, sorted, and thresholded to the maximum number of results requested. This sorted list is returned to the client.

V. Evaluation

To evaluate our proposed encrypted phrase searching and ranking search engine we indexed our encrypted modified search engine and an unencrypted standard OpenFTS/MySQL Full-Text Search database with over 500,000 emails. Numerous queries were run to compare the two search engines for speed, bandwidth, size, and security. It is expected that our methods would add certain overhead to all of the benchmarks.

A. Dataset and Experiment Setup

For our sample dataset we used a collection of over 500K emails from the former Enron Corporation that were made public by the Federal Energy Regulatory Commission during their investigation of Enron in 2003. This plaintext email archive contains over 1.8 GB's worth of data. We formatted the data to be compatible with our indexer and removed attachments, non-text words such as XML tags, and binary data. The email archive we used, along with a comprehensive description of the dataset, can be found at http://www.cs.cmu.edu/~enron/.

We used two commodity computers to conduct the experiments. One computer behaves as the client-side server, the other computer behaves as the cloud server. Each computer was installed with Ubuntu 11.10, MySQL 5.1.58, Perl 5.12.4, and OpenFTS 0.40, as well as the custom built software described in Sections III & IV. These two computers are connected through Ethernet LAN. We used a Linux utility IP Network Monitor, *IPTraf*, to capture the network traffic between the client-side server and the cloud server.

B. Speed

For our search speed experiments we eliminated the network overhead, lag, and transfer times associated with communicating across a WAN and analyze these aspects in Section V-C. Instead, we only calculate the time it takes for the cloud server to run its queries and return with the results. Furthermore, to eliminate external factors such as initial index caching and set-up times all searches are run on a hot database.

The speed of individual search queries varies proportionally to the number of documents containing the set of keywords in the query and the number of keywords in the query. The later is due to a custom MySQL user-defined function we wrote that scans each document for each keyword and location data of interest and discards the rest. This function is necessary as otherwise the database would return the entire document to the client-server, however, this user-defined function is in most cases the bottleneck in terms of time and can take up to 6 times as long as the database query itself. Research is being conducted into improving the performance of this function. Fig. 2 shows this relationship for queries contained in 100 documents to 22,000 documents. Most reasonably descriptive queries are returned in well under 10 seconds. It should be noted that many extremely common words such as

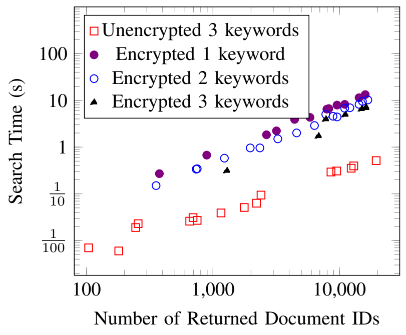

Fig. 2: The search time vs. the number of documents returned with a corpus size of 517,214 documents and β=12 bits. Search time is proportional to the number of documents returned and the number of keywords searched for.

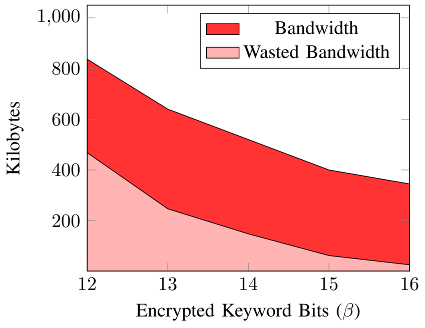

Fig. 3: The average bandwidth used for an average three word search phrase. For each additional bit in the index word roughly 21% less bandwidth is used. Wasted bandwidth is the bandwidth used to transfer documents that are eventually filtered.

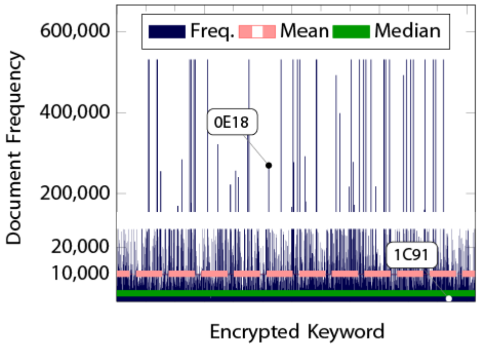

Fig. 4: Number of documents that contain the truncated encrypted keyword (β=13). Each line corresponds to a single encrypted keyword. Note: a few outliers dominate the set. An analysis attack might conclude that these encrypted keywords likely correspond to common words.

conjunctions and pronouns are included in our stop-word set and thus not included in the index.

A prohibitive part of extremely large search returns is the bandwidth considerations of transferring the results to the client-side server to be ranked. Since documents cannot be excluded prior to being decrypted the server must either arbitrarily trim its result set (thus risking filtering a document the user was actually searching for) or return megabytes worth of data to be filtered by the client-side server. Server trimming and suggesting the user to create more descriptive search queries are compromises that we could implement to handle the problem gracefully.

C. *Bandwidth Usage*

In the encrypted method bandwidth is dependent on the number of documents returned, the number of query keywords, and the number of instances the query words appear in those documents. However, a comparable unencrypted version is only dependent on the number of documents returned. For example, ‘stock’ appears often in the documents in our testing dataset. It is located in 14,828 out of the 517,214 emails in the dataset and appears a total of 63,316 times for an average of 4.27 times per email. However, another 1,412 documents contain words that encrypt to the same bits as ‘stock’ when truncated. These additional documents are returned as well. The bandwidth required to search for ‘stock’ in our encrypted database is roughly 1.19MB. ‘Dear’, on the other hand, is a word that typically begins letters but appears rarer elsewhere. It is contained in 20,093 emails but only 1.12 times per email. In addition, 1,076 other documents have keywords that collide with ‘dear’. Therefore, the bandwidth needed to search for ‘dear’ (1.07MB) is less than that of ‘stock’ even though there are 5,265 more emails with ‘dear’ in them. In contrast, both of these search terms lie almost exactly on the linear regression line for the comparable unencrypted searches and require 1.6 to 3.8 times less bandwidth than their encrypted counterparts.

A considerable amount of bandwidth transferred is due to the salting of the AES encrypted blocks. In fact, since each keyword’s location and truncated keyword index is salted individually and each salt is 8 bytes plus an attached header, between 25% and 50% of the bandwidth is due to the salt. By reducing the size of the salt or removing the header altogether this overhead can be greatly reduced while the former sacrifices security, the later’s only consequence is slightly more complexity on the client-side server for decryption.

The bandwidth needed to perform a search varies by the number of keywords specified. Fig. 7 compares bandwidth usage for searches between one and three keywords. While the increase in the number of keywords used does increase the bandwidth by roughly 90% per keyword, this affect is countered by the fact that adding keywords substantially decreases the number of documents returned. For example, to return 10,048 documents with one keyword a relatively uncommon word ‘green’ was chosen. However, to return roughly the same number of documents with four keywords, four of the most common words in the English language had to be used (‘make’, ‘time’, ‘first’, ‘like’). Each of these words are between 10 to 50 times more common than ‘green’[21].

D. *Database Size*

The database size is the last portion of our model that we analyze. It is expected that the encrypted index adds overhead compared to a non-encrypted index, which has a high rate of compression. These compression techniques are rendered useless due to the randomized nature of the encryption algorithms in our encrypted search. In addition, our index size is expected to be much larger than previous work that used Bloom filters and database indexes without storing location information. Bloom filter sizes are typically a function of the number of unique words in a document. Our inverted index is also a function of the unique words; however, the size of our non-inverted index is proportional to the total word count.

Fig. 5 compares the overall storage size for our encrypted database and a standard MySQL Full-Text search database indexed with the same documents but unencrypted. The encrypted index is larger but by a relatively small amount. For a database size of 2.6GB our encrypted solution is only 1.15GB larger than its unencrypted counterpart.

Direct comparison with other encrypted searching schemes is difficult as much of the previous work either did not publish results of the size of their system or focused on much smaller sample corpus sizes than ours, often with less than 5,000

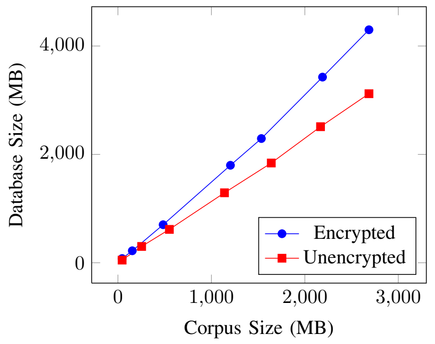

Fig. 5: Relationship between the size of the unencrypted corpus of data plus index and the encrypted corpus plus encrypted index

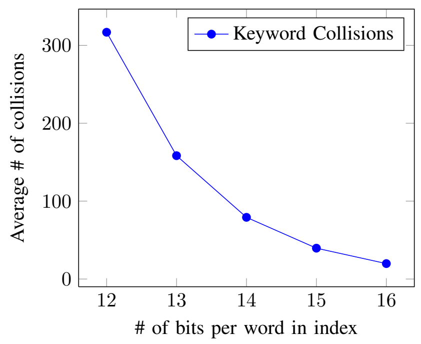

Fig. 6: Relationship between the number of bits used to encrypt words and the average number of collisions between encrypted keywords

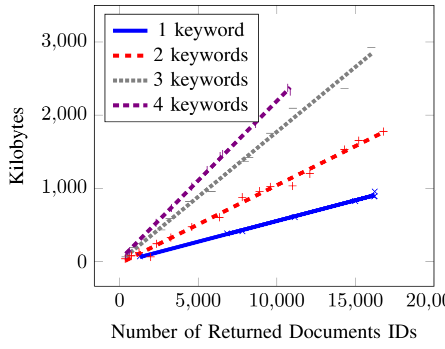

Fig. 7: Bandwidth needed for 12-bit truncated keywords with variable number of keywords in the search query. The trend-line for each group is shown.

documents [9]. However, to further show the claim that our approach is efficient, Fig. 5 shows that the size needed to store both the unencrypted and encrypted databases and original documents scale linearly with the corpus size.

E. Evaluation of the Keyword Truncated Index

Our proposed method still has areas that could be attacked. Even though there is a many-to-one relationship between keywords and their encrypted counterparts not all encrypted values are equally distributed. For example, in a 13-bit keyword truncated index, the encrypted keyword '0E18' is contained in 262,941 documents (roughly half of the corpus) while '1C91' is in only 1,005 documents. This is not all that surprising as the extremely common word, *enron*, encrypts to '0E18' while a much rarer words (such as *jackal*, among others) encrypt to '1C91'. Despite that multiple keywords encrypt to the same values of '0E18' and '1C91', a large size for the index value suggests that a common word may map to this value (Fig. 4). For larger truncated bit-values this disparity typically only increases.

As previously explained, the more bits used for the encrypted keyword the stronger an attack on the index might be. In addition, the more bits used for encrypted keywords also reduces the number of collisions (Fig. 6). However, using more bits for encryption greatly decreases the bandwidth and, thus, the speed of searches (Fig. 3). Therefore, there is an inverse relation between the security of the database and speed of the searches. It is important to understand the dataset being used and to choose a sufficient β such that the collisions sufficiently interrupt any brute-force or analysis attacks.

VI. Conclusion

We propose a unique and novel encrypted keyword searching design with proximity ranking and phrase searches within the cloud. We design a four step solution that safeguards the documents contents while still allowing for efficient searching and ranking of results that is a function of the number of unique keywords. We have developed a fully functional prototype that can work within the confines real-world response times on a large-scale dataset. In practice we show that the size overhead of the encrypted indexed database is a linear function of a non-encrypted index. Finally, whereas non-encrypted proximity ranking models have a roughly static bandwidth usage per search, our model is dependent on the number of documents with the query keyword set.

References

[1] S. Artzi, A. Kieżum *et al.*, "Encrypted Keyword Search in a Distributed Storage System," MIT, Tech. Rep. 1738, Feb. 2006.
[2] A. Aviv, M. Locasto *et al.*, "SSARES: Secure Searchable Automated Remote Email Storage," in *Computer Security Applications Conference*, Dec. 2007, pp. 129 –139.
[3] M. Raykova, B. Vo *et al.*, "Secure Anonymous Database Search," in *ACM Workshop on Cloud Computing Security*, 2009, pp. 115–126.
[4] C. Wang, N. Cao *et al.*, "Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1467–1479, 2012.
[5] E. Grosse, "Ensuring Your Information is Safe Online," http://googleblog.blogspot.com/2011/06/ensuring-your-information-is-safe.html, June 2011.
[6] S. O. Entertainment, "Sony Online Entertainment Announces Theft of Data From Its Systems [Press Release]," http://www.soe.com/securityupdate/pressrelease.vm, May 2011.
[7] D. X. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," in *IEEE Symposium on Security and Privacy*, vol. 3, 2000, pp. 44–55.
[8] S. M. Bellovin and W. R. Cheswick, "Privacy-Enhanced Searches Using Encrypted Bloom Filters," Department of Computer Science, Columbia University, Tech. Rep., Sep. 2004.
[9] C. Wang, N. Cao *et al.*, "Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data," in *IEEE International Conference on Computer Communications*, Apr. 2011.
[10] ——, "Secure Ranked Keyword Search over Encrypted Cloud Data," in *Int'l Conference on Distributed Computing Systems*, 2010, pp. 253–262.
[11] C. Raiciu, D. Rosenblum *et al.*, "Enabling Confidentiality in Content-Based Publish/Subscribe Infrastructures," in *Proc. of the 2nd Int. Conf. on Security and Privacy in Communication Networks*, Aug. 2006.
[12] M. Srivatsa and L. Liu, "Securing Publish-Subscribe Overlay Services with EventGuard," in *ACM Conference on Computer and Communications Security*, 2005, pp. 289–298.
[13] D. Boneh, G. D. Crescenzo *et al.*, "Public Key Encryption with Keyword Search," in *EUROCRYPT*, 2004, pp. 506–522.
[14] J. Baek, R. Safavi-Naini *et al.*, "Public Key Encryption with Keyword Search Revisited," in *Computational Science and Its Applications*, 2008.
[15] B. Zhang and F. Zhang, "An Efficient Public Key Encryption with Conjunctive-subset Keywords Search," *Journal of Network Computer Applications*, pp. 262–267, 2011.
[16] J. Katz, A. Sahai, and B. Waters, "Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products," in *EUROCRYPT*, 2008, pp. 146–162.
[17] E. Shen, E. Shi, and B. Waters, "Predicate Privacy in Encryption Systems," in *Theory of Cryptography Conference*, 2009, pp. 457–473.
[18] P. van Liesdonk, S. Sedghi *et al.*, "Computationally Efficient Searchable Symmetric Encryption," in *Secure Data Management*, 2010, pp. 87–100.
[19] C. Clarke, G. Cormack *et al.*, "Relevance Ranking For One to Three Term Queries," *Inf. Process. Manage.*, vol. 36, pp. 291–311, Jan. 2000.
[20] "OpenFTS," http://openfts.sourceforge.net.
[21] J.-B. Michel, S. Pinker *et al.*, "Quantitative Analysis of Culture Using Millions of Digitized Books," *Science*, vol. 331, pp. 176–182, 2011.