

Searching on Encrypted Data

Mehmet Ucal
Department of Electrical and Computer Engineering
University of Maryland College Park, MD
August, 2005
mehmet@mail.umd.edu

Abstract

In today's technology-driven world, people increasingly depend on data storage servers such as mail servers and file servers to store their data. In order to reduce security and privacy risks, the data stored on these servers should be in encrypted form and should be retrievable without loss of confidentiality. In this research project, we first conduct an extensive literature survey on the existing solutions to this problem where we have observed a challenging tradeoff between memory overhead, search capability and security when storing data in encrypted form. We then propose several new practical methods to achieve an improved tradeoff over the existing solution. These new methods are demonstrated using the ENRON email collection. Finally, we develop an interface to enable users to securely search on encrypted data with a relatively small storage overhead.

Table of Contents

1. Introduction.....	3
2. Basic Framework.....	4
3. Related Work.....	5
3.1. Keyword Based Schemes.....	5
3.2. General Word-Based Scheme.....	9
3.3. Implementation of the Non-Keyword Based Scheme.....	11
4. Proposed Improvements.....	12
5. Results and Discussion.....	14
6. Conclusion and Future Work.....	17
7. Acknowledgements.....	17
8. References.....	18

1. Introduction

In recent years, with the increased possibility of data loss and insufficient memory space on local machines, storing data on a remote database server has become a necessity for many people. For example, people usually depend on the mail services provided by a third party. In this situation the remote server, such as a mail server must be fully trusted by the user and should not reveal any information without providing proper authorization. However, in many applications, there are several factors that hinder the privacy and security features of a server. One of these factors is that the system administrators have read access to the server's system. Since in most cases the user does not know this person, hence his trustworthiness, a password-based protection will not be secure enough for the user. This factor alone should force the user to take more effective precautionary measures for protecting the stored data. Also in most applications, the data may not be stored on a single server but may be spread out on several servers, which increases the probability of an attack. Thus, if the user wants to keep the information stored on the server private, the data must be encrypted and then stored on the server.

Searching capability is one of the most useful and essential applications that many users want to perform on the data which is stored on a remote server. More specifically, the user should be able to search for a particular query and retrieve the relevant documents corresponding to the query from the database. As we have discussed before, data needs to be stored in encrypted form to secure it from malicious attackers. One obvious but naïve solution for performing the search would be to download the whole database on the local machine, decrypt all its contents, and then perform the search on the plain text. Clearly, this would require a lot of bandwidth and would be inefficient to repeat for every search and therefore not practical. Thus, there is a need to develop a solution enabling the user to perform the search over the encrypted domain in such a way that the server does not learn any unauthorized information by performing the search. The objective of this solution would be to come up with a solution that minimizes the computation on both server and user ends, and that reduces communication overhead along with memory storage at the server end.

When designing a solution for the problem of searching on encrypted data, one of the primary goals is to lower the computation at the user end since he/she is paying for the services of the server. Here, the server would take the majority of the work in terms of computation and performing the search. Another goal is to reduce the encrypted file size on the server. However, we should be aware of the fact that as the encrypted file size decreases, the number of output bits from the cryptographic encryption scheme reduce, and therefore lowering the randomness and the security of the scheme. Moreover, an elaborate and full search capability might require word-by-word encryption, resulting in increased memory usage. Therefore, in order to develop an effective solution, one should take into consideration the tradeoffs between memory overhead, search capability and security when searching on encrypted data.

To our best knowledge, the first practical solution for the problem of searching on encrypted data was introduced by Song et al. in [10]. In their sequential cryptographic

scheme, the documents are encrypted word by word using the Searchable Symmetric Key Encryption (SSKE) scheme so that the server has the capability to search over the encrypted document and determine if the word exists or not in that specific document. Song et al. provide theoretical solutions both for keyword and non-keyword based schemes. Since the non-keyword based scheme requires a more sophisticated construction compared to the keyword based scheme, their discussion focuses more on the former. The implementation of the solution for the non-keyword based scheme (two versions of the implementation are shown in Section 3.3 and in [3]) shows that this scheme still suffers from large memory overhead. This approach and the implementation of it will be discussed in detail in Sections 3.2 and 3.3.

The general word-based approach introduced in [10], provided a basis to keyword based schemes that are discussed by Boneh et al. in [2] and Goh in [6]. These schemes use the keyword approach to support a faster searching functionality. In these schemes, the user has a secure keyword list and searches the database for one of the keywords on this list. Here, even though the memory overhead is less compared to non-keyword based scheme, the keyword list limits the searching capability of the user. The properties and the variations of this approach will be discussed in detail in Section 3.1.

In this paper, we introduce a solution for the problem of searching on encrypted data that optimally combines the advantages of both the keyword based and the non-keyword based schemes. The basic idea behind our solution is to create a secure keyword list, encrypt non-keywords in the documents as a block (instead of encrypting every word separately as it was done in previous schemes), identify keywords in the documents and encrypt them separately, and make use of the keyword-based schemes' properties to perform a faster search with comparatively small overhead.

The paper is structured as follows. First, we introduce the basic framework for the problem of searching on encrypted data in Section 2. Then, we discuss the keyword based, general word-based solutions and the implementation of the non-keyword based scheme in Section 3. We describe our solution and introduce our proposed improvements in Section 4. The results of our solution and comparisons with keyword-based and non-keyword based schemes are provided in Section 5. Finally, we conclude with Section 6 and, Section 7, which discusses the future work for the specified problem.

2. Basic Framework

The basic framework is provided for the reader to have a better understanding of the encryption and search process in a typical setting. Let's assume a user, Alice, has a certain amount of documents and wants to store them on an untrusted server, Bob. For example, Alice may be a mobile user with a low-bandwidth connection who wants to securely store her email messages on Bob. Since Bob is untrusted, Alice has to encrypt her documents and only store the resulting ciphertext on Bob. For encryption, Alice uses a secret key, K , only known to her and not to Bob. After storing her documents, Alice wants to retrieve the documents containing the word W . In order to perform an efficient search for W , she encrypts W using the secret key K and sends the encrypted result along with a trapdoor for W to Bob. A trapdoor is a one-way function $f: X \rightarrow Y$ with the

additional property that given some extra information (called trapdoor information and in this case it is provided by the secret key, K) it becomes feasible to find for any $y \in \text{Im}(f)$, an $x \in X$ such that $f(x) = y$ [8]. After determining the relevant encrypted documents, Alice decrypts by using the same secret key K . This scenario, along with the necessary steps discussed that are needed for encryption and query search, is depicted in Figure 1.

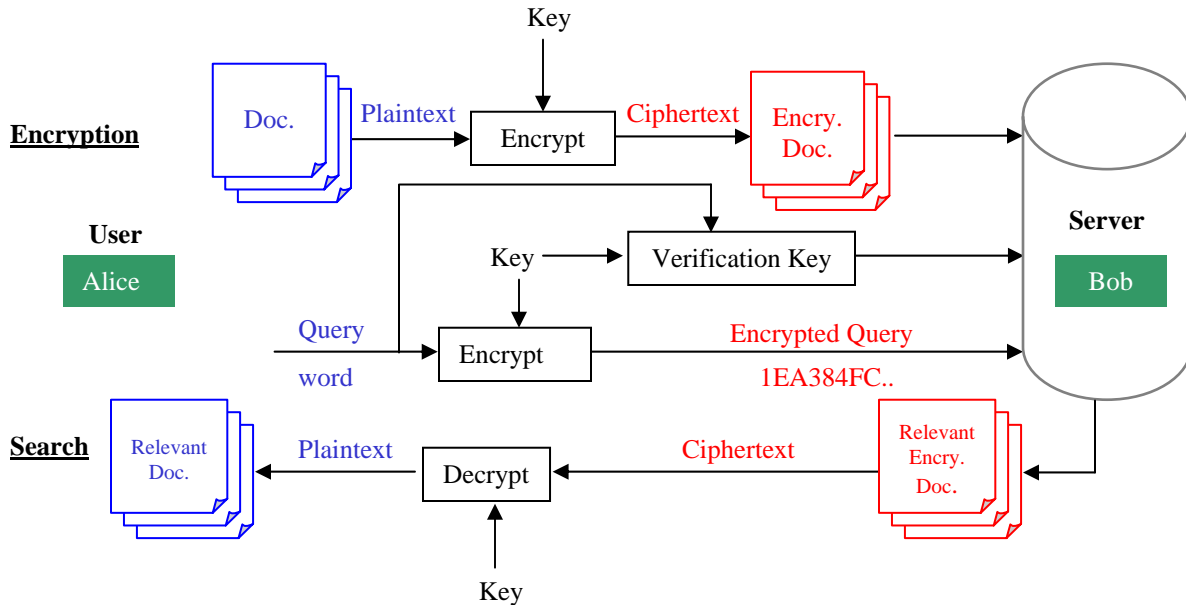


Figure 1 Basic Framework

3. Related Work

We provide discussions for three main cryptographic solutions regarding the problem of searching on encrypted data [2][6][10]. It should be noted that solutions discussed in [6] and [10] are concerned with the privacy of private databases. In private databases, a user stores its private data on a remote data storage server and later wishes to retrieve all documents containing a specific keyword from this server. In order to diminish the privacy and security risks, the data stored by the user on the remote server should be in encrypted form. Solutions in [6] and [10] provide cryptographic schemes to search on encrypted data in a private database setting.

It is worth mentioning that a public database is another important part regarding the issue of privacy of databases. In this case, a user wishes to retrieve documents from a public database without revealing any information to the database administrator. Public Information Retrieval protocols enable users to achieve this functionality in a public database setting [7]. The model discussed in [2] does not belong to either of the two settings discussed above but it is flexible enough to be extended to the required setting.

3.1. Keyword Based Schemes

There are two main schemes using the keyword-based approach. First, we discuss the solution provided in [2]. In their paper, the authors focus on a slightly different kind

of scenario than the one discussed in Section 2. In this case, a user Carl (not an untrusted server) sends another user Alice an email under Alice’s public key. Here, both the contents of the email and the keywords are encrypted. Candidates for keywords may be words on the subject line and sender’s email address. When Alice wants to search for a keyword, let’s say “urgent”, in her incoming email, an email gateway should have the ability search for the keyword “urgent” in the incoming email but should not learn anything more than the existence of this keyword in the email. Here, incoming data is from third parties making it impossible for the user to organize it (different from private database case). Also, the data is not public so PIR solutions are not relevant (different from public database case). In this case, the objective is to allow email gateway to prioritize the incoming email according to their keywords.

In order for Carl to send his email, he first encrypts his message under Alice’s public key and then appends to this ciphertext a Public-Key Encryption with Keyword Search (PEKS) of each keyword. It should be noted that PEKS is public key equivalent of SSKE discussed in [10] and in Section 3.2. Here is an example of what Carl may send to Alice:

$$E_{A_{\text{pub}}}(\text{M}) \parallel \text{PEKS}(A_{\text{pub}}, W_1) \parallel \dots \parallel \text{PEKS}(A_{\text{pub}}, W_m)$$

Here, A_{pub} is Alice’s public key, and M is the email body. Carl appends to the encrypted message the PEKS of each keyword W_i so that Alice can just give a trapdoor, T_{w_i} , to the gateway enabling her to search for the keyword W_i in Carl’s sent message. More specifically, if the gateway has $\text{PEKS}(A_{\text{pub}}, W')$ and T_w (given by Alice), it can test whether the desired W (what Alice is looking for) is equal to W' . If it is not equal, the gateway does not learn any other information. If $W=W'$, the gateway sends the relevant emails back to Alice. The authors define such a system as non-interactive public key encryption with keyword search. Such a system consists of four polynomial time randomized algorithms:

- 1) *KeyGen*(s): Input is a security parameter, s , and output is the public/private key pair $(A_{\text{pub}}, A_{\text{priv}})$.
- 2) *PEKS* (A_{pub}, W): This algorithm generates a searchable encryption for W .
- 3) *Trapdoor* (A_{pub}, W): Inputs are Alice’s public key and a word W , and the output is a trapdoor T_w .
- 4) *Test* (A_{pub}, S, T_w): This algorithm gets Alice’s public key, a searchable encryption $S=\text{PEKS}(A_{\text{pub}}, W')$ and a trapdoor T_w . It outputs ‘yes’ if $W=W'$ and ‘no’ otherwise.

So, the process works as follows: Alice generates A_{pub} and A_{priv} using *KeyGen*. She uses *Trapdoor* to produce T_w for the words of interest. Mail gateway gets T_w , A_{pub} and Carl’s keywords and uses the *Test* algorithm to determine if there is a match between W and W' .

Regarding the security of PEKS, the authors define the security semantically. For $\text{PEKS}(A_{\text{pub}}, W)$ to be useful, it should not reveal any information about W unless T_w is available. Additionally, they define an active attacker’s (A) advantage as

$$\text{Adv}_A(s) = \left| \Pr[b=b'] - \frac{1}{2} \right|$$

and say that PEKS is semantically secure against an adaptive chosen keyword attack if $\text{Adv}_A(s)$ is a negligible function.

There are two PEKS constructions provided: construction using bilinear maps and a limited construction using any trapdoor permutation. Both of these constructions are based on recent constructions of Identity Based Encryption (IBE) systems. PEKS and IBE are closely related. Boneh et al. in [2] show that constructing PEKS is a harder problem than constructing an IBE system.

Construction of PEKS using bilinear maps is based on a variant of the Computational Diffie-Hellman problem [8]. A non-interactive searchable encryption scheme (PEKS) can be built using a computable, bilinear, and non-degenerate map. With the addition of two hash functions, it is possible to build PEKS with the four polynomial time randomized algorithms discussed above. This scheme is also semantically secure and its proof of security relies on the difficulty of the Bilinear Diffie-Hellman (BDH) problem. Boneh et al. go on to prove that the PEKS based on BDH is semantically secure against a chosen keyword attack in the random oracle model assuming BDH is intractable.

The second type of construction is based on general trapdoor permutations but this type of construction is less efficient compared to first construction discussed above. Boneh et al. define a source indistinguishable (SI) system as a family of semantically secure encryptions where given a ciphertext it is computationally hard to say which public key this ciphertext is associated with. That is to say, in terms of the attacker's (A's) advantage in winning the game, a public key encryption scheme is SI if for any polynomial time attacker (A), we have Adv_{SI} as a negligible function.

Boneh et al. provide two constructions from trapdoor permutations. The first one is a simple PEKS using a keyword family, Σ . This scheme's drawback is that public key length grows linearly with Σ . Thus, the second construction allows the user to reuse individual public keys for different keywords. This feature reduces the public key size. Both of these PEKS constructions are semantically secure as long as the public key underlying encryption scheme is SI and semantically secure.

The second solution based on the keyword-based approach is provided by Goh in [6]. In his paper, the authors introduce the concept of a secure index. This secure index will only allow the user to search for a word W in the index if and only if the user has a valid trapdoor for W . The index does not reveal any information about its contents without a valid trapdoor. This scheme has a similar framework to the one that was provided in Section 2. There will be an index generated for each document on the remote server. When Alice wants to search for a word W , the *SearchIndex* algorithm below will be invoked to test for a match. Then, all the matching documents will be returned to Alice. Similar to [2], Goh's index scheme consists of four algorithms. These are:

- 1) *Keygen*(s): Input is a security parameter, s , and output is the master private key K_{priv} .
- 2) *Trapdoor*(K_{priv}, W): Takes K_{priv} and word W as input and outputs the trapdoor T_w .
- 3) *BuildIndex*(D, K_{priv}): Inputs are the document D and K_{priv} , and output is the index I_D .

4) $SearchIndex(T_w, I_D)$: Given T_w and I_D , outputs 1 if W is in the document D and 0 otherwise.

It should be noted that except the *KeyGen* algorithm, which is a randomized algorithm [4], the other three are deterministic algorithms.

In order to track words in each document, Goh uses a Bloom filter as an index. A Bloom filter is a set $S = \{s_1, \dots, s_n\}$ with n elements. The filter uses r independent hash functions h_1, \dots, h_r to determine if s is an element of S or not. If $s \in S$, then array bits at positions $h_1(s), \dots, h_r(s)$ are set to 1. In this scheme the construction of index works as follows: Given a security parameter s , K_{priv} is generated using a pseudorandom function f where $K_{priv} = (k_1, \dots, k_r) \in \{0,1\}^{sr}$, T_w is generated for the word W_m building the index using K_{priv} and D , and finally searching the index with T_w and I_D . Building the index is a two part process: For each word W_i , we compute $x_i = f(W_i, k_i)$ and then find $I_{Di} = f(D_i, x_i)$ to obtain the vector index I_D . Also, searching the index is a two part process: First we get the input x_i , compute $y_i = f(D_i, x_i)$ and then check if I_D is 1 in all y_i .

Goh's scheme uses the existing cryptographic schemes to generate the algorithm. Also, this approach can be used with compression schemes. Additionally, it is flexible to be modified for occurrence searches and conjunctive searches, which means that it can perform a search for Boolean queries "AND" and "OR" involving multiple words. This scheme is useful and it is better to be used in multiple user settings where the indexes and documents are frequently updated on the server. If just one user uses the indexes or if they are not updated regularly, the user should use a local index.

Both [2] and [6] can be categorized as keyword based schemes. The flowchart of a typical keyword based scheme is provided in Figure 2. Generally, these schemes have a small memory overhead and they are faster when compared to non-keyword based schemes. The search time is of the order of the number of keywords. On the other hand, having a keyword list limits the search capability of the user and requires updates when a document or a keyword is changed, added or deleted on the remote server.

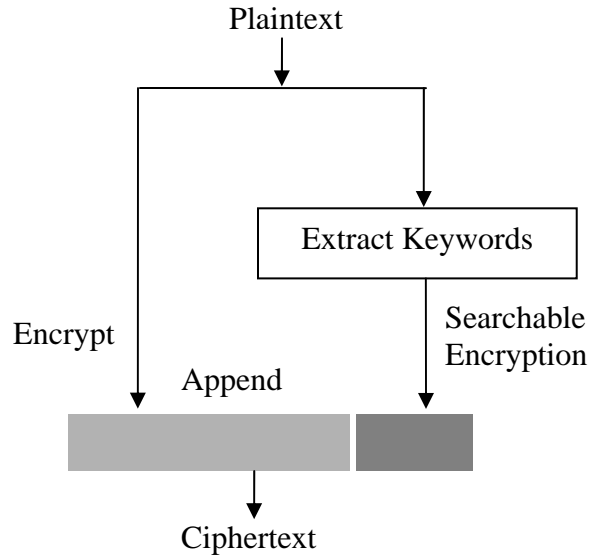


Figure 2 Keyword Based Scheme

3.2. General Word-Based Scheme

In order to discuss the proposed cryptographic scheme we start with the framework introduced in Section 2. Again, Alice wishes to retrieve documents containing the word W from Bob.

There are two types of approaches provided in [10]: performing a sequential scan without an index (non-keyword based) and performing a search with an encrypted index. First, we consider the non-keyword based approach. Song et al. start by introducing a basic scheme, which provides provable secrecy. Another name for provable secrecy is “reductionist security” meaning that one provides a reduction of the security of the scheme to the security of some underlying primitive [1]. This means that as long as the underlying primitives are secure, then the scheme, which is composed of these primitives, will be secure. Then, three configurations of this scheme are provided to achieve controlled searching, support for hidden searches and query isolation, respectively. The schemes and the properties that they achieve are discussed below.

Scheme I (Basic Scheme): It should be noted that we treat each document as a series of keyword blocks (W_1, \dots, W_L) for an L -word document). Using a pseudorandom generator G , Alice creates pseudorandom values S_1, \dots, S_L , where each S_i is $n-m$ bits long. To encrypt an n -bit word W_i , Alice concatenates S_i with $F_{k_i}(S_i)$ to get T_i , that is $T_i := \langle S_i, F_{k_i}(S_i) \rangle$. Here F is a pseudorandom function, which has an input S_i with key k_i . Finally, we get the ciphertext C_i by XORing W_i with T_i , namely $C_i := W_i \oplus T_i$. After storing the ciphertext on Bob, Alice can search for the word W by sending W and the k_i to Bob. Then, Bob XORs C_i with W_i and checks if it is in the form $\langle s, F_{k_i}(s) \rangle$ for some s . If it is in this form, it returns the result to Alice. Note that here, Bob learns the associated plaintext if k_i is given to him by Alice. This basic scheme provides provable secrecy as long as F and G are secure.

Scheme II (Controlled Searching): Here, another pseudorandom function f is introduced. In order to achieve controlled searching, the keys k_i should be chosen in a specific way so that Bob learns nothing if $W_i \neq W$. More specifically, Alice should choose

keys k_i as $k_i := f_{k'}(W_i)$, where k' is chosen uniformly randomly. By knowing $f_{k'}(W)$ and W , Bob can search for and identify the locations where W occurs ($W_i = W$), but learns nothing about the document if $W_i \neq W$. This scheme fixes one of the problems of the basic scheme.

Scheme III (Support for Hidden Searches): If Alice does not want to reveal the word W she is searching for to Bob, she can simply pre-encrypt each word W in the document using a deterministic encryption algorithm $E_{k'}$, i.e., a block cipher. It is important to note that $E_{k'}(x)$ depends only on x , it does not depend on the position of x in the document. So, if we let $X_i := E_{k'}(W_i)$, Alice will get the sequence X_1, \dots, X_L . Then, she gets the ciphertext by $C_i := X_i \oplus T_i$, where $X_i := E_{k'}(W_i)$ and $T_i := \langle S_i, F_{k_i}(X_i) \rangle$. Thus, to search for a word W , Alice just calculates $X := E_{k'}(W)$ and $k := f_{k'}(X)$ and sends $\langle X, k \rangle$ to Bob. This approach satisfies the hidden search feature if the encryption algorithm E is secure.

Scheme IV (The Final Scheme): Instead of generating keys k_i as $k_i := f_{k'}(E_{k'}(W_i))$, Alice should get her keys in a different manner because with this method, Alice has no way of knowing $E_{k'}(W_i)$ before she is able to decrypt. Proposed solution for this problem is splitting the pre-encrypted word X_i into two parts, namely $X_i = \langle L_i, R_i \rangle$ where L_i denotes the first $n-m$ bits and R_i denotes the last m bits. Now, Alice can generate k_i as $k_i := f_{k'}(L_i)$. Alice can recover L_i by XORing S_i (she knows S_i since she knows the seed) against the first $n-m$ bits of the ciphertext C_i . And from L_i , she can calculate k_i and can recover the plaintext. This scheme provides provable secrecy and query isolation (revealing a single k_i does not reveal any information other than the positions of W_i) among other features discussed above. The flow chart of the final scheme is provided in Figure 3. It should be noted that this scheme also works for words with variable lengths. This can be done by concatenating the length of the word with the word itself and encrypting this as a block. Then we can use the scheme discussed above for performing searches on the encrypted data.

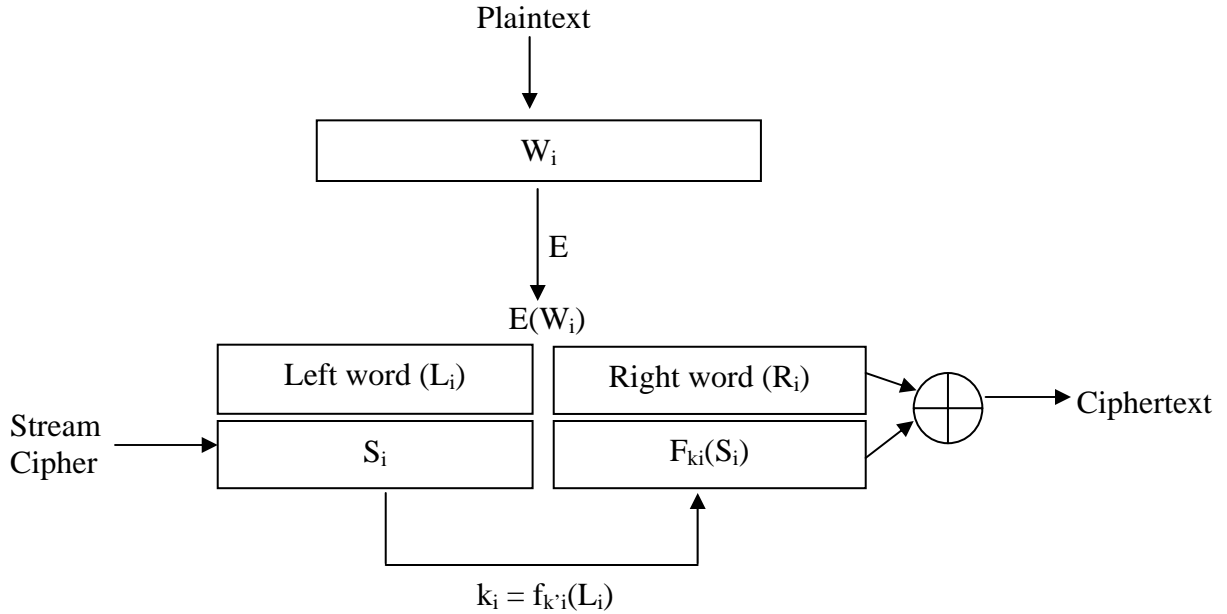


Figure 3 The Final Scheme

The second approach is searching with an encrypted index. An index consists of a list of keywords along with pointers that point to documents where the keyword appears. The keywords that are in the index are words that Alice may search for in the future. An easy but a time consuming way to search with an index is encrypting both the keywords and the pointers in the index. So, Bob can search for $E(W)$ and return the encrypted list to Alice and Alice can decrypt this list to see the document entries. Then, Alice has to send Bob a request to retrieve the documents that she needs. As we can see, this approach needs two roundtrips between the user and the server.

Another approach using an encrypted index is encrypting the list of pointers with a key k_w where $k_w = f_{k'}(E(W))$. So to search for W , Alice just sends $\langle E(W), k_w \rangle$ to Bob. Bob can do statistical analysis if the list of pointers is not a fixed size. Thus, to prevent statistical attacks from Bob, Alice should keep the list of pointers in a fixed size.

3.3. Implementation of the Non-Keyword Based Scheme

After reviewing the theoretical non-keyword based scheme discussed in Section 3.2, we describe an implementation of their theory in this section. In the previous section, there were three functions introduced in the final scheme. These were namely, E (a block cipher), f (a pseudorandom function) and F (a pseudorandom function) functions. In our implementation, we chose AES for all three functions and developed a scheme using C++ programming language. We chose AES because it satisfies all of the security requirements that were presented in [10] and it offers a variety of different modes for encryption and pseudorandom number generation. For our implementation, E is AES in CBC mode, and F and f functions are AES-generated pseudorandom functions. Since AES takes a 128-bit block as an input, each word W_i in the document must have a 16-character (128 bits) length but X_i has to be 256 bits. The reason for this length is the concatenation of S_i and $F_{k_i}(S_i)$, both 128 bits in length, to get T_i . Then, T_i has to be 256 bits and we cannot XOR T_i and X_i without having the same length to get C_i (Obviously, C_i will be 256 bits as well). Thus, this forces us to use 256 bits for the length of X_i . Having a fixed word length means that we pad the shorter words and split the longer words so that we have a fixed length for each of the words in the document. In our implementation, we have one master key (seed for S), which also has a length of 128 bits, and all the other keys are derived from this master key. It should be emphasized that the lengths discussed in this section are specifically for AES-type cryptographic schemes. These numbers would appropriately change if another scheme is used instead of AES as will be shown in Section 4.

For our dataset, we chose to use the ENRON Email Corpus. It contains data from about 150 users, mostly senior management of Enron, organized into folders. The corpus contains a total of about 0.5M messages [5].

After running simulations for numerous email messages from this dataset, our results can be summarized as follows:

- Encryption time is linear in document size.
- There is a substantially large memory overhead.
- Search time is of the order of the number of words in the document.

Experimental data gathered in [3] show very similar conclusions to ours.

After reviewing and implementing this cryptographic scheme, we are now able to discuss the advantages and the disadvantages of this scheme in a broader sense. As we have discussed before this scheme is provably secure and it supports controlled searching, hidden queries, and query isolation. There are several advantages to sequential scan. First of all, the user has the capability to search for any word in the document and to find the location of the word and the number of times the word occurs in the document. The document updates can be done very easily using this scheme. Also, key management and key storage is convenient since there is a key hierarchy in place (all keys can be generated from one master key, making key management easy for Alice).

One of the problems with the non-keyword based scheme is that there is a large memory overhead. If we want to reduce the storage space, we have to reduce the encrypted word length, but this would cause a reduction in the degree of the security. Therefore, there is a tradeoff between storage space and security. Another problem is that the work is linear in document size, meaning that the larger the dataset the larger the encryption and search times. Also, this scheme needs various modifications to handle variable length words making this scheme impractical at times. Additionally, if Alice allows Bob to search for too many words, Bob will be able to do statistical attacks. In order to prevent such attacks Alice has to change the key frequently, re-encrypt the documents with this new key, and change the order of the ciphertext with some pseudorandom permutation. The degree of security reduces dramatically when Alice wants to perform complex search queries such as conjunctive searches. For example, when performing such a query, here let's say Boolean query for W_1 "AND" W_2 , Bob performs a search for the documents containing W_1 and performs another search for W_2 . Then, Bob finds the documents that were in both of these searches and sends them to Alice. But optimally, Bob just needs to know the documents containing both W_1 and W_2 together. If Bob knows more than this information with a certain number of conjunctive searches, it would be very easy for Bob to do statistical attacks. Thus, using this scheme for conjunctive searches poses a serious risk for the user.

Using an index is much faster in terms of search time if we have documents that are large in size. Also, performing a search with an encrypted index needs less sophisticated constructions than non-keyword based approach. However, index-based schemes suffer from insecure updates. For example, if Alice changes her documents she has to update the index. If she adds a new document and does not update the list of pointers for one keyword entry, Bob would be able to tell that the keyword does not appear in the new document that was just added. Also, as we have discussed above there are several other ways for Bob to do statistical attacks when using an index-based scheme.

4. Proposed Improvements

The implementation of non-keyword based scheme in Section 3.3 raised several interesting issues such as the effectiveness of encrypting each word W_i as a block, and the large memory overhead. In order to improve the existing scheme, we incorporate

some of the advantages offered by the keyword-based schemes. As we discussed in Section 3.1, keyword based schemes have very small overhead when compared to non-keyword based schemes. Also, their search time is much lower.

In this section, we propose several improvements to the scheme discussed in the previous section in order to increase performance in encryption time, search time, and storage space. Our modifications are based on the observations that we have made in our implementation in the previous section along with the benefits offered by the keyword-based schemes when used over large datasets. Our intention is to optimally combine the advantages of both schemes (keyword and non-keyword) to get better performance and storage results.

One of the observations that we made in the previous section was that the size of the encrypted text was very large. The reason for this was the concatenation of S_i and $F_{k_i}(S_i)$, both 128 bits in length, to get T_i (256 bits in length). Then, the resulting ciphertext C_i had to be 256 bits. In this case, we used AES for pseudorandom function F . One of the proposed modifications is, instead of using AES for F , we should use a keyed hashing function such as the Message Authenticator Algorithm (MAA) [9]. MAA is a hashing algorithm that takes n bits (n can be any number) as input along with a 64-bit key and outputs a 32-bit result. So, when this algorithm is integrated into our scheme, it will take S_i (in this case, AES will generate S_i as a 96-bit pseudorandom number) and will output a 32-bit result. So when concatenated, the resulting T_i will be 128 bits in length. Hence, the length of the ciphertext C_i will be cut in half if a hashing algorithm is used for F . However, this scheme will surely have a reduced degree of security since we use a 96-bit pseudorandom number instead of the original 128 bits (the randomness is reduced). Also, MAA is not as secure as AES [8]. So, the overall security of the scheme will be diminished. But in many applications, the benefits of this modification, such as reduced storage space and faster encryption times, outweigh the risks posed by the lower levels of security. As indicated in the previous sections, we have a tradeoff between memory overhead and security. While the original scheme is more secure it occupies more memory in contrary to our proposed improvement. Thus, our scheme would achieve a different tradeoff point, which might be very helpful for practical purposes.

Another observation made in the previous section was that each word W_i in the document has to have a 16-character (128 bits) length. This means that no matter what the length of the word is, it is always padded so that the length is 128 bits. Most words in the English language are less than 10 characters meaning that usually more than half of the space in W_i is padded. Our objective is to use this space efficiently so that we are able to encrypt more words in less time. In order to achieve this, we incorporate the keyword-based approach into our scheme. In most practical applications, the users are more likely to search for some specific words (keywords). Also, our initial experiments showed that the keyword presence in a document will be no more than 10%. Therefore, we propose the user to create a secure keyword list and encryption to be done according to this list. So, we first identify keywords and non-keywords in a document by using the keyword list, and then we combine non-keywords together until the 16-character length is reached and send this block (with more than multiple words along with appropriate spaces in it) for encryption. All keywords in the documents are identified and encrypted separately.

By doing so, we can achieve our objective to decrease encryption time and memory overhead. It should be noted that by integrating the keyword-based scheme, we also lose the privilege to search for any word in the document. The user is only allowed to search for the words that are in the keyword list. Also, the proposed scheme requires updates when a document or a keyword is changed, added or deleted on the remote server. It should be noted that these observations were made in Section 3.1 regarding the traditional keyword based approaches.

For searching over email data, it is important a scheme has the capability to perform field search. Most of the times the user wants to search certain fields for the emails received on a specific date or wants to retrieve emails sent by a specific person. Without this capability, the user will have to do some extra work to find what he/she is really looking for. Field search capability allows the user to search for a specific part of the email and retrieve the emails of interest more efficiently. Our user-friendly scheme can search for the following fields in an email: Date, From, To, Subject, and Body Text. We achieve this capability by the structure that is common to all of the emails in the dataset.

These modifications provide the user many additional advantages when compared to the original scheme discussed in the previous section. The modified scheme has less memory overhead and is much faster than performing a linear search. Also, this scheme has several advantages over the keyword-based schemes. It can find the location of the keyword in the document, it can determine the number of occurrences of the keyword in the document, and it can obtain statistics and other relevance details. None of these capabilities were present in the previously discussed keyword-based schemes (Section 3.1).

5. Results and Discussion

In this section, we integrate the proposed improvements to our existing implementation discussed in Section 3.3 and discuss the results. As it was the case in the previous implementation, our email dataset is ENRON Email Corpus and our algorithm was developed in C++ programming language. Even though our proposed scheme has the capability of searching over different fields, for a fair comparison, the keyword search is done only over the body text of the email messages in our simulations.

After running several simulations and observing the results, we see that our results comply with our theoretical predictions from the previous section. Table 1 and Figure 4 show that in terms of the expansion factor, the proposed scheme has much lower values than the keyword-based scheme. We remind the reader that keyword based schemes have lower memory overhead when compared to non-keyword schemes. Thus, surely the difference, in terms of the memory overhead, between the proposed scheme and the non-keyword based scheme will be much bigger.

Table 1 Comparison of Proposed Scheme and Keyword Scheme

Ratio of keywords to total number of words	Number of documents (Total 100)	Expansion factor	
		Proposed scheme	Keyword based
0.0000 — 0.0401	4	1.2887	1.3251
0.0401 — 0.0802	42	1.3355	1.3567
0.0802 — 0.1203	27	1.2630	1.3172
0.1203 — 0.1604	19	1.2236	1.2533
0.1604 — 0.2406	2	1.1966	1.2145
0.2406 — 0.2807	3	1.4558	1.6222
0.2807 — 0.4010	3	1.9285	2.5945

Expansion Factor

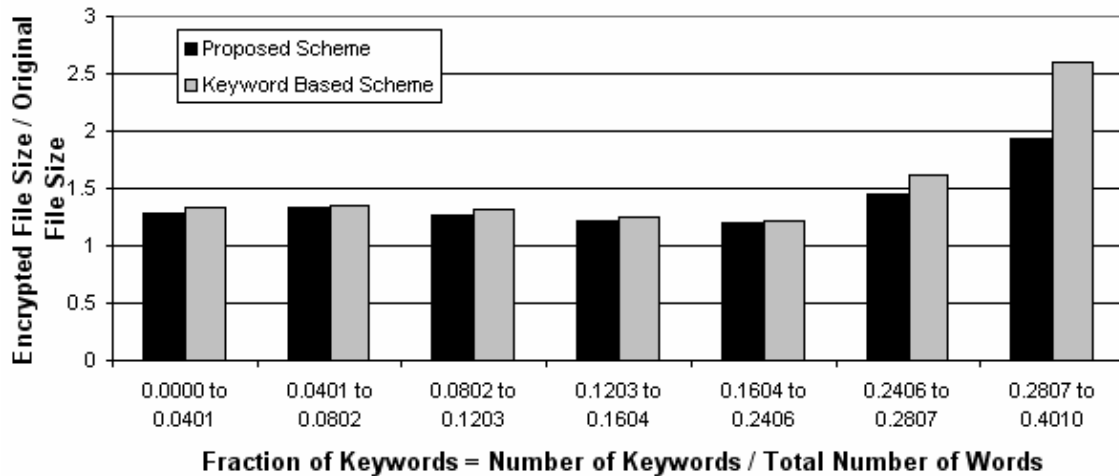


Figure 4 Expansion Factor for Enron Email Corpus

The expansion factor is equal to:

$$\text{Expansion Factor} = \frac{\text{File Size of the Encrypted Document}}{\text{File Size of the Plain Text}}$$

Also, the relationship between the memory overhead and the expansion factor is given by the formula:

$$\text{Memory Overhead (in percentage)} = (\text{Expansion Factor} - 1) \times 100$$

Putting it simply, memory overhead is the additional work done to produce the encrypted text document. From Table 1, it can be seen that the keywords do not comprise more than 40% of a document, which is an expected fraction. Also, we see that most of the documents contain keywords around 10% of the whole document. It should be emphasized that the proposed scheme produces much better results no matter what fraction of keywords make up the document. From this table, we conclude that the proposed scheme has lower storage overhead when compared to keyword-based schemes and non-keyword based schemes.

The difference, in terms of the expansion factor, between the proposed scheme and the keyword-based scheme is even more evident in Figure 5. This figure shows the expansion factor of all three schemes in terms of the number of keywords present in the document. As one can see from the graph, the difference between keyword based scheme and proposed scheme becomes larger as the keyword number increases in a document. The constant line in the figure represents the non-keyword based scheme. Since this scheme performs the encryption for every word in the document, it is not relevant how many keywords are present in the document. This scheme treats all the words as keywords and encrypts each of them by padding to a predetermined word length. Thus, the expansion factor for the non-keyword based scheme does not change with the number of keywords in the document.

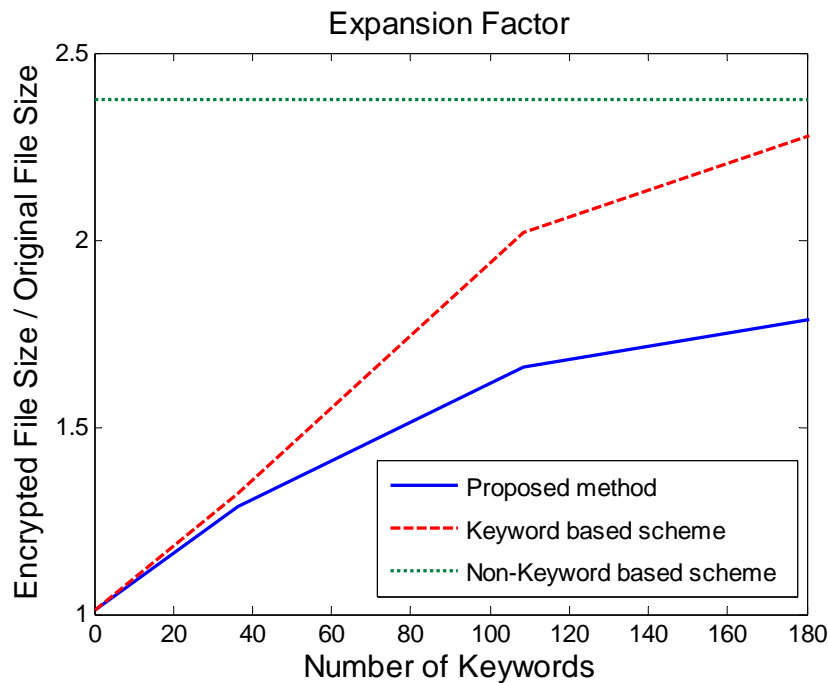


Figure 5 Comparisons of Proposed Method, Keyword and Non-Keyword Based Schemes

We observe that from the improvements proposed in the last section, we have greatly improved the original scheme discussed in Section 3.3. With the integration of the keyed hashing function MAA, the proposed scheme has a lower encryption time when compared to the non-keyword based scheme. It has a lower storage overhead. Also, it has the capability of searching over various fields in an email message making it very

convenient for the user to perform searches on the encrypted data. Therefore, the proposed method achieves the overall objective of creating a better tradeoff between search functionality, memory overhead and security.

6. Conclusion and Future Work

In this paper, we consider the problem of searching on encrypted data. We present an extensive literature survey on this subject and provide detailed analyses for the existing solutions; keyword and non-keyword based approaches. We identify memory overhead as one of the problems and suggest some practical modifications for reducing it. We propose to extend the existing non-keyword based scheme by using a keyed hashing function. Furthermore, we show that we can substantially reduce the memory overhead by combing the non-keywords in the document. These improvements are motivated by the practical applications and optimally combine the advantages of both the keyword and non-keyword based techniques. We implement the proposed improvements on the ENRON Email Corpus and show that the effectiveness of the proposed scheme is greater than the existing schemes, especially in terms of the memory overhead and encryption time. We conclude that the proposed scheme achieves the objective of improving the tradeoff between memory overhead, search capability and security when storing data in encrypted form.

Although the proposed improvements greatly increase the quality of the user's search capability, there is, still, room for improvement in terms of the scalability of the schemes. Also, the incorporation of the keyword list into our scheme limits the capability of the number of searchable words. The next objective should be to provide the user a better way to update the document and keyword list, and offer the freedom to search for any word in the document.

7. Acknowledgements

I would like to thank Dr. Min Wu for help and guidance with this project. I would also like to acknowledge the kind help and input of Ashwin Swaminathan, Yinian Mao, and Shan He throughout this project. This work was supported by the University of Maryland MERIT 2005 Program.

8. References

- [1] M. Bellare, “Practice-oriented provable-security,” In *Proceedings of International Workshop on Information Security (ISW 97)*. Springer-Verlag, 1998. Lecture Notes in Computer Science No. 1396.
- [2] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, “Public-key encryption with keyword search,” In C. Cachin, editor, *Proceedings of Eurocrypt 2004*, LNCS. Springer-Verlag, May 2004.
- [3] R. Brinkman, L. Feng, S. Etalle, P. H. Hartel, and W. Jonker, “Experimenting with linear search in encrypted data,” Technical report TR-CTIT-03-43, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, Sep 2003. <http://www.ub.utwente.nl/webdocs/ctit/1/000000d9.pdf>.
- [4] T. Cormen, C. Leiserson, and R. Rivest, “Introduction to Algorithms”, MIT Press, 2000.
- [5] Enron Email Dataset. See <http://www.cs.cmu.edu/~enron>
- [6] E-J. Goh, “Secure indexes,” in Cryptology ePrint Archive: Report 2003/216, February 2004. See <http://eprint.iacr.org/2003/216/>.
- [7] E. Kushilevitz and R. Ostrovsky, *Replication is not needed: Single Computationally-Private Information Retrieval*, in FOCS 97.
- [8] A. Menezes, P. van Oorschot and S. Vanstone, “Handbook of Applied Cryptography,” CRC Press, 1996.
- [9] B. Preneel, P. van Oorschot, “On the Security of Iterated Message Authentication Codes,” in *IEEE Transactions on Information Theory*, Vol. 45, No.1, Jan 1999.
- [10] D. X. Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.