|  |
| --- |
|  |
| Encrypted Searching |
| Literature Review |

|  |
| --- |
| Hiroshi Fujinoki, Alexander Towell  3-5-2014 |

Contents

[2 Confidentiality 2](#_Toc381659672)

[2.1 Compression 3](#_Toc381659673)

[2.2 Obfuscation 3](#_Toc381659674)

[2.3 Encryption 4](#_Toc381659675)

[2.3.1 Symmetric Encryption 4](#_Toc381659676)

[2.3.2 Public-key encryption (asymmetric) 4](#_Toc381659677)

[2.3.3 Cryptographic One-Way Hash Functions 4](#_Toc381659678)

[3 Types of information leaks 4](#_Toc381659679)

[3.1 Document (data) confidentiality 5](#_Toc381659680)

[3.2 Query privacy 5](#_Toc381659681)

[3.3 Access patterns 5](#_Toc381659682)

[3.4 Primary goal of encrypted search schemes 6](#_Toc381659683)

[4 Online and offline searching 6](#_Toc381659684)

[4.1 Online searching 7](#_Toc381659685)

[4.2 Offline (index) searching 7](#_Toc381659686)

[4.2.1 Inverted index 7](#_Toc381659687)

[4.2.2 Bloom Filter (Signature File) 8](#_Toc381659688)

[4.2.3 Perfect Hash + Cryptographic Hash 8](#_Toc381659689)

[5 How do we map queries to documents? 9](#_Toc381659690)

[5.1 Boolean keyword matching 9](#_Toc381659691)

[5.1.1 Extensions 10](#_Toc381659692)

[5.1.2 Exact phrase matching (word n-grams) 11](#_Toc381659693)

[5.2 Relevancy (context-aware) 12](#_Toc381659694)

[5.2.1 Types 12](#_Toc381659695)

[6 Multi-user encrypted searching 14](#_Toc381659696)

[7 References 15](#_Toc381659697)

[8 References 17](#_Toc381659698)

Related Work

In [9], one of the earlier papers presented on encrypted searching, the author observes that many individuals and organizations wish to exploit cloud storage, but do not trust cloud storage providers (CSP) with their confidential data.

The naive solution is for clients to encrypt the document on their trusted local machines, and then upload it to the untrusted CSP. Subsequently, when clients need to access the documents, they may download it to their local machines and decrypt it for viewing. However, if clients are on devices with limited resources, this is not be reasonable solution. It becomes especially unreasonable as the volume of sensitive information expands (to gigabytes or more). Rather, what is needed is some way to allow the CSP to search on behalf of clients, returning only those documents that are relevant to the user’s query, but without disclosing the contents of the data nor the contents of the query[[1]](#footnote-1) to the CSP.

The capability to search over an encrypted collection of documents without needing to first decrypt them is known as encrypted searching. It may be considered a subset of the more general (and far more computationally demanding) category of fully homomorphic encryption (FHE), which DARPA has recently spent $20 million dollars to support its development.

Encrypted searching has gained a lot of traction in the research community because of its obvious utility. Many solutions have been proposed using diverse concepts and methods. This section describes the major existing related work and serves as a general overview of its current status, especially as it relates to the strengths and weaknesses of the various proposals.

# Confidentiality

The first question to address is how should confidentiality be provided?? That is, what techniques will be used to prevent the disclosure of information to unauthorized parties, e.g., a remote server hosting the encrypted documents? We consider three primary approaches: compression, obfuscation, and encryption.

|  |  |  |
| --- | --- | --- |
| Methods | Advantages | Disadvantages |
| Compression  [13] | Very space efficient  Fast and easy to implement  Very well understood (Huffman) | To serve as an obfuscator, separate symbol mapping table must be maintained locally by users. (Why not just query this table?)  Serves as a substitution cipher; may be broken through cryptanalysis. |
| Obfuscation  [17], [19] | Effective non-skilled attackers  Compared to Huffman codes, hash tables (can) be far smaller in size | High insider risk; may have clues or be trusted with secrets.  Slow? |
| Encryption  [1], [2], [4], [5], [6], [9], [11] | Effectively protects the contents of documents (or indexes) from their release of contents from anyone (including insiders, as long as insiders cannot access decryption keys) | Depending on the confidentiality provided, performing approximate matching and ranking relevancy of documents is very difficult.  Strong encryption tends to be slow.  If note careful, still prone to information leaks, e.g., frequency analysis. See: information leaks |

## Compression

In [13], the idea of using the theoretically optimal symbol encoder, Huffman, is proposed in the context of information retrieval (without consideration of the unique needs of encrypted search), where the symbols consist of words rather than letters. Thus, each word in the document maps to a unique bit string such that the total length of the document is (near) optimally compressed. A compelling notion, to conceal the contents of documents, is to use this approach to generate code-words as a form of encryption (essentially, a substitution cipher). That means the symbol table must be kept private to prevent untrusted parties from learning the code-words.

When combined with an inverted index[[2]](#footnote-2), this is fast and efficient; it only takes log(n) time to query whether a word is in a document, where n is the size of the vocabulary. It is also very space efficient, i.e., an optimal symbol compressor is being used as the encoder.

Unfortunately, it has far more disadvantages than advantages. First, a local symbol table must be maintained and kept private. Second, since the symbol mapping must be kept private, it is reasonable to ask whether users should even bother querying the server at all when they can alternatively simply query the local private symbol table.

If the symbol table is not being optimized per document, e.g., optimal code-words for the entire document collection are used instead, then only a single symbol table must be stored and maintained for all the documents in the collection, but at the cost of less than optimal compression. However, and this brings us to the third and primary disadvantage, the symbol table—being a substitution cipher—is easy to break through basic cryptanalysis. **[Flaws of Substituion Ciphers Frequency List of English Alphabet]**

## Obfuscation

The compression section discussed a method to obfuscate by symbol remapping, or substitution. In general, any symbol substitution technique may be used to obfuscate the contents of documents. The main distinction between obfuscation and compression, in this context, is obfuscation obscures the contents of documents by mapping symbols, words, or phrases to other symbols, words or phrases [17, 19], whereas compression code-words map ASCII codes to binary codes.

Conceptually, there are two types of obfuscations: encoding-based and index-based obfuscations. For example, encoding-based obfuscations transform the appearances of symbols, words, and phrases by applying a universal transformation for a document. Thus, the word “ABC” may be transformed to “BCD” by numerically adding one to the ASCII code of each character while their data type remains unchanged (their data type is still ASCII character string). Indexed-based transformations apply a unique transformation to the provided input. For example, the index may, as an invertible function, map “John Smith” to “Person 192353” (or simply “192353” if the “Person” tag leaks too much information). Another example can be found in automatically generating hash functions, e.g., map input x to x’ and store x’ in the index. As with compression, the symbol mapping must be kept hidden from untrusted parties

These techniques suffer from the same problems that compression code-words suffer from: they are relatively easy to break or at the very least leak a lot of information about the contents.

## Encryption

Well-tested encryption standards make the most sense if the desired outcome is proven confidentiality.

In data encryption parlance, the document (information) is called plaintext. The plaintext is fed into an encryption function whose output is called cipher text. In addition, the function requires at least one more parameter called an encryption key. The idea behind the use of a key is instead of needing to generate a new algorithm to generate a different encryption, a different key can be used to the same effect. That is, a given plaintext must generate a different cipher text (the encryption function must be invertible) for each key in the key address space.

Without knowing the key, it ought to be the case that the only way to break the cipher is to do a brute force exploration over the entire key space, e.g., for a given cipher text c, apply the encryption algorithm and a key to see if that generates a plausible plaintext c’. To prevent the tractability of brute force attacks, the key space tends to be quite large, e.g., 2128.

### Symmetric Encryption

In the context of encrypted searching, two different types of encryption have been used, symmetric and asymmetric (public-key) encryption. Symmetric encryption uses a single key for both encryption and decryption. Compared to public key encryption, it is less computationally demanding. However, the downside is, a secure channel must be used to communicate the secret key if multiple parties need to be able to use it. The earliest examples of encrypted search used symmetric encryption [9].

### Public-Key Encryption (Asymmetric)

One of the earlier proposed encrypted searching proposals based on public-key encryption with keyword search overcomes a weakness in symmetric encrypted searching: to send a searchable encrypted document to Alice, you may use her public key such that only she can search it using her corresponding private key.

### One-Way Cryptographic Hash Functions

In a secure offline index-based data structure (see offline searching), which is independent of the document it represents, there is no need (nor desire) to be able to losslessly reconstruct the document from the information in the index. This provides us with another attractive option: do not use encryption at all, but rather, use a one-way (non-invertible) cryptographic hash function. [7]

They may still be combined with one or more secret keys, as with symmetric and asymmetric encryption, to manage user authorization.

There are a couple of advantages to this approach. First and foremost, it is far less computationally demanding. Second, because it is non-invertible, even if the secret key (or keys) are revealed, this does not necessarily compromise the contents of the actual document (except by allowing whatever searching facilities the offline index permits).

# Types of information leaks

In [7], the author contends that an encrypted search capability should reveal no information about the contents of the document unless a secret (called a trapdoor) is known which allows the secret-holder to search it is known. The only information that should be revealed through this search operation is whether a given encrypted document is relevant to a query. Thus, even if an untrusted party—like an untrusted cloud storage provider— captures an encrypted query (the trapdoor), it can neither determine the contents of the query nor the document, affording both data confidentiality and query privacy.

While this is ideal, to enable encrypted searching in untrusted environments, there are many different, sometimes subtle, ways in which information can be disclosed—or “leaked”— to untrusted parties. This remains true even if we assume a strong, unbreakable encryption cipher is being used.

## Document (data) confidentiality

Consider two distinct documents, A and B. If the same term (or word) in document A and document B has same representation (in the index), this constitutes an information leak. At this point, it is essentially a substitution cipher; in theory, not only do these leak information, but they can be broken through cryptanalysis.

Indeed, even if the same term maps to the same representation in different parts of the same document, this also constitutes a substitution cipher. This is one of the reasons why Goh [7] argues that traditional hashes are not suitable for use as a secure index.

## Query privacy

A similar argument to data confidentiality applies to query privacy. In the extreme case, queries may be sent in plaintext. This will be very informative to an untrusted party; over time they may be able to construct a model of an encrypted document by submitting many different plaintext queries to it and seeing if it is relevant or not relevant, e.g., does it contain these keywords in the case of a Boolean keyword query?

A weaker variation of this is when a query for a specific keyword or term always looks the same when submitted to any document in the collection. In this case, through statistical inference, an untrusted party can slowly build up frequencies for patterns in the secure indexes. If they combine this with a bit of prior knowledge, like the frequency distribution of English words, they may be able to figure out what each pattern is most likely to represent (e.g., likelihood maximization using Naïve Bayes).

A simple solution is to, for instance, insert a cryptographic hash of the term concatenated with a secret plus the document’s unique ID. In this way, the same query on different documents will look completely different. However, as elaborated on elsewhere, this imposes serious challenges to the efficiency of relevancy (context-aware) scoring.

## Access patterns

To exacerbate matters, even if an encrypted search scheme provides robust data confidentiality and query privacy, implicit information useful for statistical inference may still be leaked. For example, if a private query consisting of some random sequence of bits is frequently followed by another action, like checking stock prices, this constitutes a correlation which, conceivably, may be used to infer properties about the query and the corresponding documents that are returned in response to it.

To mitigate this more subtle form of information disclosure, Pinkas **[Oblivious RAM Revisited]** proposed the use of oblivious RAM constructs to conceal the history of data access patterns (observable by the server) associated with a user’s encrypted searching activities.

Oblivious RAM may be thought of in the following way: to prevent meaningful statistical inferences from being made about a user’s activities, whenever an action—a read or write—is performed, randomly include other randomly chosen reads and writes (or queries) to obscure what the user was actually interested in.Unfortunately, the overhead cost of this may not yet be practical, but it is one of the only ways to prevent this subtler form of information leak.

## Primary goal of encrypted search schemes

In light of these types of information leaks, the primary goal of encrypted searching is to prevent untrusted parties from inferring anything about the encrypted document beyond which documents are returned for a given encrypted query.

Most encryption searching schemes [7][9][Public-key encryption with keyword search][Confidentiality-Preserving Rank-Ordered Search][…] claimed this to be their guiding principle, although only a few solutions, like that proposed by Pinkas [Oblivious RAM], considered maintaining this confidentiality standard when confronted with an untrusted party who considers the history of the user’s activities.

# Online and offline searching

| Methods | Advantages | Disadvantages |
| --- | --- | --- |
| On-line  [1], [5], [6], [9] | Exact phrase matching is easy—does not blow up size like in other solutions  Approximate matching (see section 3) is easier to implement, but it is still problematic given the fact that exact matches on encrypted bit strings must be performed | Words and phrases may be statistically inferred by observing the frequency of encrypted token patterns.  It is vulnerable to substitution cipher attacks  Sequential search; slow and impractical for large-scale use |
| Off-line: Inverted index  [2], [13] | Extremely well-understood  Context-aware searches are easy to perform  Fast term lookups: O(log n)  Space-efficient (if Huffman compression)  Data structure facilitates efficient ranking operations (see section 3) | Potentially vulnerable to substitution cipher attacks  Potentially vulnerable to preimage collision attacks  Leaks more information than Bloom filter |
| Off-line: Bloom filter  [7][Encrypted Keyword Searching in a Distributed Storage System] | Fast term lookups: (O(1))  Very space efficient (nearly optimal)  Can trade accuracy for space-complexity.  Rapid index construction  Data structure facilitates efficient ranking operations | While it is O(1), this constant time complexity is hiding a large coefficient—k cryptographic hash functions must be evaluated, where k can become quite large.  Deleting a term from a document (a particular word does not appear in a document after the deletion) requires re-construction of Bloom filters |
| Offline: Minimum perfect hash  [Network Applications of Bloom Filters: A Survey] | Fast term lookups (O(1))  Extremely space efficient (nearly optimal)  Unlike Bloom filter, only two hash functions to evaluate.  Can trade accuracy for space-complexity  Data structure facilitates efficient ranking operations | Leaks more information than Bloom filter indexes.  More complicated construction (and slightly slower, although linear in the number of the members still) |

## Online searching

On-line search performs a sequential search on the document cipher [1, 5, 6, 9, *Public-key encryption with keyword search*]. To be able to perform encrypted searches on such a cipher, a block cipher may not be used; rather, each term must be encrypted separately to facilitate exact string matches on the encrypted query terms. This is a simple substitution cipher; as mentioned elsewhere, these may be broken, or at the very least, they leak a significant amount of information.

Moreover, as pointed out in **[Confidentiality-Preserving Rank-Ordered]**, proposals based on online searching, in light of their sequential time complexity, are not appropriate except in limited contexts. For example, they may be appropriate if the purpose is to allow an email server to obliviously scan the “subject” field of incoming emails for the keyword “urgent” and, if detected, the recipient is immediately notified.

While their disadvantages are many, they do have some advantages. Their primary advantage is related to their simplicity: one simply iterates through all of the terms and applies a cryptographic hash to each. To permit exact phrase searching, an encrypted query phrase need only be iteratively compared to the encrypted terms in the cipher. A comprehensive overview of online searching solutions can be found in [5].

## Offline (index) searching

Offline indexes are data structures which store a representation of the document (or documents) in which rapid, efficient retrieval and ranking operations are facilitated. They also have the added benefit that, in many cases, they are completely isolated from the document they are representing, and thus both can be designed and implemented separately and independently.

For example, they may use independent encryption and compression algorithms, appropriate to their specific needs. Furthermore, they can be independently distributed.

In light of these clear advantages, most of the recently proposed encrypted searching constructions are based on offline indexes [2][7][**Encrypted Keyword Search in a Distributed Storage System**] like Bloom filters.

A comprehensive overview of offline-based solutions can be found in [13].

### Inverted index

In [Adding Compression to Block Addressing Inverted Indexes], a possible approach to a secure index is elaborated upon. Previously, we discussed this index in the context of using Huffman codes to serve as a substitution cipher. However, other, more secure solutions are possible.

If terms, or words, are being stored in the inverted index, then this means an untrusted party may observe the contents of the document directly, and so no strong data confidentiality is provided. Moreover, even if encrypted, compressed or obfuscated transformations of terms are being stored in the index, it is still potentially vulnerable to cryptanalysis (e.g., frequency estimates may be estimated. Alternatively, you could try various words to try to do a preimage collision attack on the values in the index. However, if multiple values may map to the same value in the index, this does complicate frequency analysis – at the expense of permitting false positives.

The primary advantage of the inverted index is that it is extremely well-understood (it is the most popular index in the field of information retrieval at large), fast, and space-efficient (especially if a block-level or document-level postings list is used).

### Bloom Filter (Signature File)

A Bloom filter [?][?] is a probabilistic (approximate) set which can trade accuracy for space complexity. It consists of a bit vector of size m, all initially set to 0, and k (traditionally cryptographic and independent) hash functions. For each member, use the k cryptographic hash functions to map it to k (or fewer, since collisions are possible) positions in the bit vector, setting each of those bits to 1. To check if an element is a member of the set, check to see if each of its k hashes positions are set to 1. If any are still 0, then it is definitely not a member (no false negatives). However, if all of them are set to 1, we may assume that it is a member. However, it is possible, with some probability, that it is not a member; rather, one or more actual members caused those k bit positions to be set to 1. Thus, Bloom filters allow for a tunable false positive rate.

It is fairly straight-forward to construct a secure searchable index from a Bloom filter. For each term (words, or n-grams, or other constructions) in the document, insert it into the filter. To prevent unauthorized users from querying the index, do not insert the plaintext terms; rather, insert some transformation of them. Ideally, use a one-way hash function, or a cryptographic hash function, on the term concatenated with a secret, e.g., insert(bloom, hash(terma | secret)). To further harden the Bloom filter from cryptanalysis, e.g., to guard against information leaks possible through correlation analysis, the same terms in separate documents ought to map to different index positions in the Bloom filter. In [7], it is recommended that the document id be appended to the ciphertext terms of the document during the construction of the secure index. For example, insert(bloom, hash2(hash1(terma | secret) | doc\_id). Likewise, during the construction of encrypted (hidden) queries, apply the same transformations. Since *secret* is unknown to untrusted parties, like the server, they are unable to query the secure index, and since the doc\_id is different for each document, term regularities never materialize.

The primary advantage of the Bloom filter, in the context of encrypted searching, is that it leaks very little information. In a Bloom filter, not only is it possible for more than one term to map to the same positions, but mappings for different terms overlap and thus one cannot know if a pattern of ones is from a combination of n terms or a single term. This is why, according to [7], the Bloom filter is such a strong choice. Additionally, the Bloom filter is also space efficient, since we may freely trade accuracy (false positives) for space-complexity (number of bit positions in the filter).

Bloom filters have two notable weaknesses. First, Bloom filters do not identify the position (which is both a blessing and a curse in the context of encrypted search) of the terms (of the encrypted document) inserted into the set; this complicates approximate matching and context-aware searching, which we will elaborate on in the section “How do we map queries to documents.” Second, Bloom filters (with the exception of Bloom filter variations, like Counting Bloom [**An Improved Construction for Counting Bloom Filters**]) do not allow removal of members from the set, thus necessitating the Bloom filter’s reconstruction whenever removal is required.

### Perfect Hash + Cryptographic Hash

As promising as the Bloom filter is, it is not as space efficient as a minimum perfect hash, which may also be used to trade accuracy for space-complexity, but it does so in a theoretically optimal way. In addition, the minimum perfect hash construction also allows for the evaluation of only two hash functions instead of k, thus it has a time complexity advantage over Bloom filters which may need to evaluate ten or more hash functions.

The primary disadvantage, in the context of encrypted searching, is that it may leak, for a given false positive rate, more information than the Bloom filter index since none of the members collide with each other in a minimum perfect hash—that is why it is called a perfect hash. In addition, the time to construct a minimum perfect hash is potentially slower (although it is still linear in the size of the members) than with the Bloom filter.

As a demonstration, here is a possible approach to making a minimum perfect hash allow for false positives [**Network Applications of Bloom Filters: A Survey**]:

* membership\_vector[perfect\_hash(x)] = crypto\_hash(x)
* To check if an x’ is a member, see if membership\_vector[perfect\_hash(x’)] equals crypto\_hash(x’).
* If x’ is really not a member, then it will test positively as a member with probability 0.5number\_of\_bits\_output\_by\_cryptographic\_hash.

It should be noted that, while in theory the perfect hash is optimal with respect to bits per member for a given false positive rate, implementations in practice do not achieve this limit. In practice, the space-complexity savings over the Bloom filter is less significant, although the time complexity is a big selling point.

# How do we map queries to documents?

The question of how to effectively map queries to documents is a very important, often neglected topic, in the Encrypted searching community. There are two primary ways being explored in the encrypted searching community: Boolean keyword matching and context-aware (degrees of relevancy) searching.

|  |  |  |
| --- | --- | --- |
| Methods | Advantages | Disadvantages |
| Boolean keyword search (n-gram search)  [1], [6], [9], [20], [21], [22], [23], [24], [25] | Simple  Can easily be extended to complex searches, e.g., a disjunction of terms.  Can be extended to support fuzzy set membership queries more easily than relevancy scoring. | May return unrelated search results due to lack of quantifying relevance to given query  If using online method, it will result in high run-time overhead for large documents and/or large collections. |
| Context-aware search  [10], [12], [18], [26], [27], [29] | Much better precision and recall on results  Draws from extensive research in the IR community | Run-time search overhead, but can be managed. Searches using secure indexes are more complicated |

## Boolean keyword matching

Boolean keyword searches [Public-key encryption with keyword search, Practical Techniques for Searches on Encrypted Data] look for ~~a~~ particular words (or more generally, terms). Although the number of words being looked for varies from 1 to a particular number, what is common in this approach is that they perform a Boolean test regarding whether the given set of words appear in a document. This type of search does not keep track of anything other than the Boolean test – either all of the words matched, or none of them did. And while there are some techniques that are tolerant to “deviations”—like typographical errors—encrypted search must still rely on some form of exact string matching due to both the terms in the query and the document being encrypted. For example, the solution proposed by Li [6] addresses tolerance of typographical errors by including all error patterns up to k errors directly in the index, upon which simple exact matching may be performed.

### Extensions

#### Conjuctive keyword search

In [1], they propose a system which permits secure conjunctive queries for certain keywords (trapdoors) on a given set of fields, like the “From” field in an email. By secure, they mean that given access to a set of indexes for encrypted documents and a freely chosen set of trapdoors, adversaries—like an untrusted cloud storage provider—must not be able to learn anything about the encrypted documents except whether it matches those specific trapdoors.

Their work demonstrates an improvement over the single keyword searching discussed in [9], but their solution is still rather limited. They still only perform exact string matching (instead of approximate string matching), their solution inflexibly requires the document creator to tag specific keyword fields for search-ability—in their case, these field names (although not the actual values) are exposed, which constitutes an information leak—and finally they do not even consider untrusted parties which consider historical data (previous queries and matching results).

#### Approximate keyword matching

In [*Efficient Fuzzy Search in Large Text Collections*], the authors point out that, for Google, not returning enough results is not a problem. Their primary problem is finding ways to return fewer, more relevant results so that users do not have to sift through too many results (most users only check the first page of results from Google). Thus, Google is motivated to improve the precision, which is the ratio of relevant documents returned to the total documents returned. However, in vertical search--such as encrypted searching over an enterprise's store of encrypted documents--there is far more concern over not missing or overlooking relevant documents, since there may be so few relevant results to begin with. Thus, vertical search tends to have an objective in direct contrast with Google’s. That is, recall, which is the ratio of relevant documents return to total number of relevant documents, is equally or even important to than precision.

Elsewhere, we discuss relevancy scoring, which provides a more sophisticated approach in that the goal is to rank documents according to how relevant they are to a query as opposed to the simple Boolean "relevant" or "irrelevant" score found in Boolean keyword searching. But, a simple extension to Boolean keyword searching which improves the recall (at the expense of precision) is to do more tolerant matching on the keywords.

##### Locality-sensitive hashing

In locality sensitive hashing [Distance-Sensitive Bloom Filters], the notion is to map similar (according to some distance measure) items to the same hash. This is an especially good fit in the context of encrypted searching, since in encrypted searching, only exact matches (on the encrypted bit strings) is possible.

To avoid the curse of dimensionality—or in other cases avoid having to explore a space that combinatorially explodes—in [Approximate Nearest Neighbors: Towards Removing the Cuse of Dimensionality], locality-sensitive hash functions for dimensionality reduction; that is, use hash functions in which the probability of a collision is high for “close” elements and low otherwise. (Distance preserving.) Thus, LSH functions are not at all like cryptographic hash functions; cryptographic hashes, like most hash functions, are designed to minimize the probability of collisions, but in general LSH hashes are designed to maximize collisions in some sense.

For instance, in [**Locality-Sensitive Bloom Filter for Approximate Membership Query**], the authors observe that Bloom filters generally assume cryptographic hash functions, or at least hash functions which uniformly distribute over the domain (bit positions). However, what if this requirement is relaxed? Then, a choice of hash functions can be made which are more likely to test positively for non-members that look like members by using locality-sensitive hash functions. Unfortunately, this will cause more information leakage; for minimizing information leakage, the hash functions should uniformly distribute over the entire domain.

###### Stemming

Stemming may be thought of as another especially relevant form of locality sensitive-hashing. In stemming, morphological variations of a word are mapped to a single base form. By reducing such variations to a single form, in which the different variations have the same essential meaning, recall and precision can hopefully both be improved.

For example, if a user searches for “computing grades”, it would seem the user would find “computed grade” relevant also. By not including this variation in the result set, the recall and potentially the precision are reduced: the recall is reduced because not all of the relevant documents are returned, and the precision is potentially reduced because a less relevant document may be returned in its place. Stemming has demonstrated itself to be a fast and fairly effective technique to improve precision and recall. [Viewing Morphology as an Inference Process]

###### Phonetic algorithms

Phonetic algorithms are another form of locality sensitive hashing. The notion is to map words that sound alike to the same hash. Soundex is one of the more popular examples of this; it is an especially useful trick for approximate matches on the names of people.

##### Edit distance

In [Fuzzy Keyword Search Over Encrypted Data In Cloud Computing], a mechanism is proposed to address the limitation in which only exact matches on keywords are allowed. In particular, they propose a construction which allows for matches on typographical errors or spelling variations, e.g., “color” vs “colour”.

To accomplish this, when constructing the secure index, for each term in the document, add all k-edit error patterns, where an error is an insertion, deletion, or substitution of a character. For example, for a 1-edit error tolerance, the keyword “age” is expanded to {age, \*age, a\*ge, ag\*e, age\*, \*ge, a\*e, ag\*}, where the \* represents any character. Thus, if “age” fails to match, the query can be automatically expanded to each of those variations in turn until a match is found.

##### Wildcard matching

Wildcard searches can be quite useful. For example, if users are unclear on how to spell a particular word, they can use wildcards to represent their ignorance, e.g., instead of “tomorrow”, they may type “to\*row”. Or, as another example, the user may seek multiple variations of a word, e.g., “\*night” for “night” or “knight”.

The solution proposed in [Fuzzy Keyword Search Over Encrypted Data In Cloud Computing] can be readily repurposed to implement wildcard searching.

### Exact phrase matching (word n-grams)

Most searchable encryption schemes only allow matches on keywords, but in [**Phrase Search over Encrypted Data with Symmetric Encryption Scheme**], a method for secure exact phrase matching is elaborated on. Phrase searches consist of approximately 10% of web search queries, so this is an important capability. Unfortunately, they require clients maintain a local dictionary on their computers to facilitate the capability. As long as such data must be maintained locally to perform searches, one may reasonably argue that local searchable indexes should be maintained instead. Local indexes, freed from many of the security concerns, would permit any sort of search operation without the need to communicate with server until a specific document is desired (thus less information leaks in the form of access patterns)

#### Biword model

The notion is, to accomplish exact phrase queries, as long as the index supports 2-grams, any n-gram exact phrase search can be expanded to a sequence of 2-grams. For example, to find the exact phrase, “hello dr fujinoki” can be represented as the Boolean keyword query, “hello dr”, “dr fujinoki”. Do note, however, that this opens up the possibility for false positives, as this expanded query will also match any document in which “hello dr” and “dr fujinoki” are present—they do not have to be adjacent to each other.

Simple extensions can exploit k-gram members, like 3-grams, to reduce the probability of a false positive. The biword model would work well with secure indexes, like the Bloom filter or minimum hash constructions.

## Relevancy (context-aware)

As pointed out in [Confidentiality-Preserving Rank-Ordered Search], most encrypted searching research focuses on Boolean search, where a document is relevant to a query if and only if all of the terms in the query match. As indicated elsewhere, this has a number of problems with respect to precision and recall. The results in this paper represent an important advance over prior encrypted searching schemes in that it ranks documents (out of a set of documents) according to estimated relevance to a query.

In modern IR systems, scoring the relevancy of a document to a query is, arguably, its most important task. If standard relevancy scoring techniques in IR can be brought to bear on encrypted searching, the utility of encrypted search will be significantly improved.

However, encrypted searching poses a number of challenges to standard relevancy metrics in IR. In encrypted search, a server obliviously searches over a collection of encrypted documents; however, the confidentiality guarantees (query privacy and data confidentiality) complicate most of the traditional relevancy scoring techniques found in the IR literature, like tf-idf, unless query confidentiality requirements are relaxed.

For instance, if a particular term maps to a different bit string in each document (to minimize information leakage as explained elsewhere), and no information may be leaked to the server about the query terms, then the user would need to submit a separate query for each document, which would be quite costly.

On the other hand, if this requirement is relaxed, much more can be done. In particular, if query terms to always map to the same bit string representation, as seen by the server, for every document, then the CSP itself can receive a single query from the user and take it from there.

### Types

#### Weighted keywords

Weighted keywords is based on two fundamental insights. First, some of the terms occur more frequently in a document than other terms. So, when scoring the relevancy of a document, if a frequent term in the document matches a term in the query, it should be given more weight than a less frequent but matching term. Mathematically:

The second insight is that many words (or terms) will be in all, or most, of the documents in the collection. These words, therefore, carry very little meaning; they have no discriminatory power as they appear in nearly every document. Conversely, many words will be very rare or even unique in a collection, and thus they have significant discriminatory power. For example, the word “the” is in nearly every document—it serves as linguistic glue— but the word “acatalepsy” will be found in very few, if any. The heuristic, then, is the more discriminatory power a term has, the more weight it should be given when scoring a document’s relevancy. Mathematically:

Combining these two insights, we have tf-idf (term frequency, inverse document frequency) and its variants.

A secure index, like a Bloom filter, can easily be adapted to work with these metrics [Spectral Bloom Filters, Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement].

#### Term Proximity

In [Efficient Text Proximity Search], the importance of proximity of terms in keyword searches is considered. The fundamental principle can be demonstrated by considering the following: given two documents, document1 = "A B C" and document2 = "A D D ...  D B C", document1 should be more relevant than document2 for the query "A B" even though they both contain the keywords with the same frequency. Put simply, how close together are the terms? The closer, all things else being equal, the better the match.

There are two primary approaches to account for this. The first way is to implicitly model term proximity by represent the documents at finer granularities, e.g., every sequence of N sentences gets a different index. Thus, the size of N can be tuned to provide a desired level of precision and recall: the larger the N, the better the recall but the worse the precision and, likewise, the smaller the N the better the precision but the worse the recall. The second way is to explicitly introduce a proximity measure, e.g., assign larger relevancy scores to matches in which the terms are “closer”. While both approaches tend to involve rather informal, ad hoc decisions, it is necessary to make them to create an effective IR system.

#### Semantic search

In [Concept Search: Semantics Enabled Information Retrieval], the authors make it clear that every previous search technique—from simple Boolean keyword searching to tf-idf proximity weighted relevancy scoring—are variations of syntactic search in which some form of string matching, combined with various ways to judge how important particular string matches, is being performed.

There are two major problems with this string matching approach. First, different words (or phrases) may be used to express similar meanings (depending on the context). This is referred to as synonymy. And, second, the same word (or phrase) may be used to express different meanings (depending on the context). This is referred to as polysemy. Both of these problems harm the relevancy of results.

Semantic search takes a different approach. Instead, it asks, what is the meaning of the text, and does the query in some way correspond to that meaning? This is a more complicated question. It may, for instance, involve natural language processing to perform word-sense disambiguation, part of speech tagging, and named entity recognition. When combining this with ontological and semantic knowledge, the IR system may begin to process queries in a way that resembles a human's ability to understand text.

For instance, if a user asks for "carnivore hunting prey", one may assume he is also interested in more specific concepts, like "dog chasing cats" or "lions hunting antelopes". Using part of speech tagging, it can be determined that "carnivore" is the subject, "hunting" is the verb, and "prey" is the object. Using word-sense disambiguation, the word senses can be determined fairly accurately, e.g., "carnivore" maps to "carnivore-1" (word sense 1). Using an ontology (like Wordnet), it can be determined that "carnivore-1" is a concept which includes (more specific concepts) like "dog-1", "lion-2", etc. Then, it may be assumed we can expand the subject, "carnivore-1", to {"carnivore-1", "dog-1", "lion-2", ...}, the verb, "hunting-3", to {"hunting-3", "chasing-1", "preying-4", ...}, and the object "prey-4" to {"prey-4", "feline-2", "antelope-1", "cat-1", ...}. Clearly, this is not an easy problem, but this approach can already, in limited ways, be put to effective use.

There are also statistical techniques to model semantically related terms, like latent semantic indexing (LSI). Unfortunately, there has been very little progress on either of these fronts in relation to encrypted searching. They represent promising future directions.

# Multi-user encrypted searching

In [Veriﬁable Symmetric Searchable Encryption for Multiple Groups of Users], the authors observe that most encrypted search implementations assume only one person will be able to perform the searches; or, if multiple people, then they all must share the same secret, and that secret will allow them to query the secure index. However, what if one wishes to be able to revoke the ability for a user to query the secure index?  Take this example:

* User 1 makes secure index for document using secret
* User 2 is trusted to query secure index by sharing with him the secret
* User 1 no longer wants user 2 to be able to query the secure index

How can this be accomplished? One solution is to simply use two secrets, and only give User 2 one of the secrets. Example:

* User 1 makes secure index for document using secret1 and secret2
* User 2 is trusted to query secure index, partly, by sharing with him secret1
* User 1 gives a server secret2
* User 2 may submit encrypted queries to server on secure index
* Server cannot determine contents of encrypted query, nor contents of secure index, except which documents are ranked as relevant to the encrypted query
* User 2 still needs the server because both secret1 and secret2  must be used to query secure index
* User 1 no longer wants user 2 to be able to query the secure index. He informs server not to honor his query requests
* Even if User 2 has a local copy of secure index, he cannot query it since he does not know secret2 and he can’t ask the server to query it on his behalf since the server has been instructed to de-authorize him.

 In pseudo code, a term is hashed into the secure index like so (or something more elaborate):

hash2(hash1(termplain text | secret1), secret2)

Thus, to check the secure index for termplain text, both secret1 and secret2 must be known.

This is the essential idea behind multi-user systems and proxy encryption. Note that these models generally assume the server (who is trusted with one of the required secrets) and the partially trusted users do not collude. If server agrees to continue servicing requests of de-authorized users or if server gives the users secret2, then they will be able to continue querying the secure index.

# References

|  |  |
| --- | --- |
| [1] | P. Golle, J. Staddon and B. Waters, "Secure Conjunctive Keyword Search over Encrypted Data," *Applied Cryptography and Network Security, Lecture Notes in Computer Science,* vol. 3089, pp. 31-45, 2004. |
| [2] | Z. Wei, Z. Dan-Feng, G. Feng and L. Guo-Hua, "On Indexing and Information Disclosure Measure for Efficient Cryptograph Query," *Proceedings of the World Scientific and Engineering Academy and Society International Conference on Computers,* pp. 476-480, 2009. |
| [3] | Q. Liu, G. Wang and J. Wub, "Secure and Privacy Preserving Keyword Searching for Cloud Storage Services," *Journal of Network and Computer Applications,* vol. 35, no. 3, pp. 927-933, May 2012. |
| [4] | C. Dong, G. Russello and N. Dulay, "Shared and Searchable Encrypted Data for Untrusted Servers," *Data and Applications Security XXII, Lecture Notes in Computer Science,* vol. 5094, pp. 127-143, 2008. |
| [5] | M. R. Asghar, G. Russello, B. Crispo and M. Ion, "Supporting Complex Queries and Access Policies for Multi-User Encrypted Databases," *Proceedings of the ACM Workshop on Cloud Computing Security Workshop,* pp. 77-88, 2013. |
| [6] | J. Li, Q. Wang, C. Wang, N. Cao, K. Ren and W. Lou, "Fuzzy Keyword Search over Encrypted Data in Cloud Computing," in *Proceedings of IEEE INFOCOM*, 2010. |
| [7] | E.-J. Goh, "Secure Indexes," *Trust, Privacy, and Security in Digital Business, Lecture Notes in Computer Science,* vol. 3592, pp. 128-140, 2005. |
| [8] | Q. Liu, G. Wang and J. Wu, "An Efficient Privacy Preserving Keyword Search Scheme in Cloud Computing," *Proceedings of International Conference on Computational Science and Engineering,* vol. 2, pp. 715-720, August 2009. |
| [9] | D. X. Song, D. Wagner and A. Perri, "Practical Techniques for Searches on Encrypted Data," *Proceedings of the 2000 IEEE Symposium on Security and Privacy,* no. Dawn Xiaodong Song, David Wagner, and Adrian Perrig, pp. 44-55, 2000. |
| [10] | H. Cao, D. Jiang, J. Pei, E. Chen and H. Li, "Towards Context-Aware Search by Learning a Very Large Variable Length Hidden Markov Model from Search Logs," *Proceedings of the International Conference on World Wide Web,* pp. 191-200, 2009. |
| [11] | H. B. a. M. Celikik, "Efficient Fuzzy Search in Large Text Collections," *ACM Transactions on Information Systems,* vol. 31, no. 2, pp. 1-59, May 2013. |
| [12] | F. Giunchiglia, U. Kharkevich and I. Zaihrayeu, "Concept Search: Semantics Enabled Syntactic Search," *Proceedings of CEUR Workshop,* 2008. |
| [13] | G. Navarro, E. S. de Moura, M. Neubert, N. Ziviani and R. Baeza-Yates, "Adding Compression to Block Addressing Inverted Indexes," *Information Retrieval,* vol. 3, no. 1, pp. 49-77, 2007. |
| [14] | S. Pearson, Y. Shen and M. Mowbray, "A Privacy Manager for Cloud Computing," *Cloud Computing, Lecture Notes in Computer Science,* vol. 5931, pp. 90-106, 2009. |
| [15] | Y. Lu and G. Tsudik, "Enhancing Data Privacy in the Cloud," *Trust Management V, IFIP Advances in Information and Communication Technology,* vol. 358, pp. 117-132, 2011. |
| [16] | R. Latif, H. Abbas, S. Assar and Q. Ali, "Cloud Computing Risk Assessment: A Systematic Literature Review," *Future Information Technology, Lecture Notes in Electrical Engineering,* vol. 276, pp. 285-295, 2014. |
| [17] | M. Mowbray, S. Pearson and Y. Shen, "Enhancing Privacy in Cloud Computing via Policy-Based Obfuscation," *Journal of Supercomputing,* vol. 61, p. 267–291, 2012. |
| [18] | R. A. Baeza-yates, "Text retrieval: Theory and practice," *In 12th IFIP World Computer Congress,* vol. I, pp. 465-476, 1992. |

[19] Christian Collberg, Clark Thomborson, and Douglas Low, “A Taxonomy of Obfuscating Transformations,” 1997.

[20] Y. C. Chang and \_\_\_\_\_, “Privacy Preserving Keyword Searches on Remote Encrypted Data,” *Lecture Notes in Computer Science*, vol. 3531, pp. 442-455, 2005.

[21] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra, “Executing SQL over Encrypted Data in the Database-Service-Provider Model,“ *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 216-227, 2002.

[22] Christoph Bösch, Richard Brinkman, Pieter Hartel, Willem Jonker, “Conjunctive Wildcard Search over Encrypted Data,” *Proceedings of the VLDB International Conference on Secure Data Management*, pp. 114-127, 2011.

[23] Benwen Zhu, Bo Zhu, and Kui Ren, “PEKSrand: Providing Predicate Privacy in Public-Key Encryption with Keyword Search,” Proceedings of IEEE International Conference on Communications, pp. 1-6, 2011.

[24] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky, “Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions,” *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 79-88, 2006.

[25] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, “Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data,” in *Proceedings of IEEE INFOCOM*, pp. x-x, Apr. 2011.

[26] Huanhuan Cao, Derek Hao Hu, Dou Shen, Daxin Jiang, Jian-Tao Sun, Enhong Chen, and Qiang Yang, “Context-Aware Query Classification,” *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3-10, 2009.

[27] Biao Xiang, Daxin Jiang, Jian Pei, Xiaohui Sun, Enhong Chen, and Hang Li, “Context-Aware Ranking in Web Search,” *Proceeding of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 451-458, 2010.

[28] Andrei Broder and Michael Mitzenmacher, “Network Applications of Bloom Filters: A Survey*,*” *Internet Mathematics*, vol. 1, no. 4, 2002, 485-509.

[29] Yelong Shen, Jun Yan, Shuicheng, Lei Ji, Ning Liu, Zheng Chen, “Sparse Hidden-Dynamics Conditional Random Fields for User Intent Understanding,” *Proceedings of the International Conference on World Wide Web*, pp. 7-16, 2011.

[30]

[Z] Jidong Chen, Hang Guo, Wentao Wu, and Wei Wang, “iMecho: a Context-Aware Desktop Search System,” *Proceedings of the International ACM SIGIR conference on Research and development in Information Retrieval*, pp. 1269-1270, 2011.

[?] A. Swaminathan, Y. Mao, G.-M. Su, H. Gou, A. Varna, S. He, M. Wu, and D. Oard, ““Conﬁdentiality-Preserving Rank-Ordered Search”

[?] 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.

[?] Ralf, Andreas, Seungwon, Martin, and Gerhard, “Efficient Text Proximity Search”.

[?] Yinqi Tang, Dawu Gu, Ning Ding, and Hiaining Lu, “Phrase Search over Encrypted Data with Symmetric Encryption Scheme”.

[?] Michael Mitzenmacher and Adam Kirsch, “Less hashing, Same Performance: Building A Better Bloom Filter”.

[?] Michael Mitzenmacher, “Compressed Bloom Filters”.

[?] Spectral Bloom Filters

[?] Oblivious RAM Revisited

[?] Efficient Text Proximity Search

[?] Verifiable Symmetric Searchable Encryption for Multiple Groups of Users

[?] Network Applications of Bloom Filters: A Survey

[?] Distance-Sensitive Bloom Filters

[?] Extremely Fast Text Feature Extraction for Classification and Indexing

[?] Locality-Sensitive Bloom Filter for Approximate Membership Query

[?] Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement

[?] Public-key encryption with keyword search

[?] Building A Better Bloom Filter

[?] Phrase Search over Encrypted Data with Symmetric Encryption Scheme

[?] Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality

1. See section 3 for types of information leaks [↑](#footnote-ref-1)
2. See section 3.2.1 on Inverted Indexes [↑](#footnote-ref-2)