Encrypted Searching

Literature Review

Hiroshi Fujinoki, Alexander Towell

3-5-2014

1 CONTENTS

2 Purpose of Encrypted Search 2

3 Confidentiality 2

3.1 Compression 3

3.2 Obfuscation 3

3.3 Encryption 4

3.3.1 Symmetric Encryption 4

3.3.2 Public-Key Encryption (Asymmetric) 4

3.3.3 One-Way (Cryptographic) Hash Functions 5

4 Types of information leaks 5

4.1 Document confidentiality 5

4.2 Query privacy 6

4.3 Access patterns 6

4.4 Primary goal of encrypted search schemes 7

5 Online and offline searching 7

5.1 Online searching 7

5.2 Offline (index) searching 8

5.2.1 Inverted index 8

5.2.2 Bloom Filter (Signature File) 8

5.2.3 Problems with the Bloom filter 9

6 How do we map queries to documents? 10

6.1 Boolean keyword matching 10

6.1.1 Extensions 10

6.1.2 Exact phrase matching (word n-grams) 12

6.2 Relevancy (context-aware) 12

6.2.1 Types 13

7 Multi-user encrypted searching 14

8 References 15

2 Purpose of Encrypted Search

In [1], one of the earlier papers presented on encrypted searching, the author observes that many individuals and organizations wish to exploit cloud storage services, but do not trust cloud storage providers (CSP) with their sensitive 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 their documents, they may download it to their local machines and decrypt it for viewing. However, this is not an ideal option. For instance, if clients are on devices with limited resources (mostly bandwidth), it is unreasonable to expect them to download the entire set of documents, decrypt them, and then execute local searches on them. It becomes especially unreasonable as the volume of sensitive information expands.

What is needed is some way to allow the CSP to search the encrypted documents on behalf of clients, returning only those documents relevant to user queries. Furthermore, this should be done without revealing the contents of both the documents and the queries. In other words, the CSP should be able to perform oblivious searches on behalf of users.

The ability to search over an encrypted collection of documents without needing to 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) [2], which DARPA has recently spent $20 million dollars in support of its continuing development.

Encrypted searching has gained a lot of traction in the research community because of its obvious and timely utility. Many solutions to this problem have been proposed; this section provides a general overview of the field and describes much of the major existing work in the area. Particular attention is paid to the strengths and weaknesses of the various proposals.

3 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 **[2], [3]**	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 structure instead?) Serves as a substitution cipher; may be broken through cryptanalysis
Obfuscation **[4], [5], [6]**	Effective against non-skilled attackers	High insider risk; may have clues or be trusted with secrets Usually too weak for cryptographic use
Encryption **[7], [8], [9], [10], [11], [1], [12], [12]**	Assuming key is kept private, provides strong guarantee on data confidentiality. Very well understood (cryptography)	Depending on the types of information leaks prevented, certain IR operations problematic, e.g., if query privacy is provided, ranking relevancy of documents can be more challenging

		Strong encryption tends to be slow. And still prone to information leaks

3.1 Compression

In [2], the idea of using the theoretically optimal symbol encoder, Huffman [3], 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 codewords 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 codewords.

When combined with an inverted index[1], this is not only space-efficient, but fast also; it only takes *O(log n)* time to query whether a word is in a document, where n is the size of the vocabulary. 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 tables are not being optimized per document—e.g., optimal codewords for the entire document collection are generated—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 on reasonably small units—is relatively easy to break through cryptanalysis. It has too many repeating patterns (and those patterns represent actual words, in this case).

3.2 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 primary distinction between obfuscation and compression is obfuscation obscures the contents of documents by mapping vocabulary symbol sequences to other symbol sequences [4] [5] (e.g., phrases to other phrases), whereas compression maps codes in one vocabulary (e.g., character strings) to codes in another vocabulary (e.g., binary strings).

Conceptually, there are two types of obfuscation techniques: encoding-based and index-based. For example, encoding-based obfuscations transform the appearances of symbol sequences by applying a universal transformation for a document. Thus, the character sequence *A B C* may be transformed to *B C D* by numerically adding one to the ASCII code of each character (this is also an example of a Caesar substitution cipher, but in this context it is referred to as an obfuscation technique).

Indexed-based transformations apply unique transformations to each input. For example, the index may invertibly map *John Smith* to *Person 192353* (or simply *192353* if the *Person* tag leaks too much information).

1 See section 5.2.1 on Inverted Indexes

Another example can be found in automatically generating hash functions; that is, map input x to x’=hash(x) and store x’ in the searchable index. As with compression, the symbol remapping instructions—essentially a secret key—must not be disclosed to unauthorized parties.

These techniques suffer from many of the same problems that compression codewords suffer from: they can be broken (as relatively simple substitution cipher equivalences) or leak information about the obfuscated contents.

3.3 ENCRYPTION

All of the previous confidentiality techniques serve the same purpose: convert communicated information into a secret code so that unauthorized people cannot understand it. The more popular and secure approach to doing this is through cryptography.

In cryptography, the unencrypted data is called plaintext. To encrypt (convert) plaintext into an unintelligible format, called ciphertext, the plaintext is input into an encryption function along with a cryptographic key. The inverse of the encryption function is the decryption function, which takes the ciphertext and a cryptographic key (either the same key, as in symmetric encryption, or another paired keyed, as in asymmetric encryption), and outputs the plaintext. For each cryptographic key in the keyspace (all possible keys), the encryption function maps a given plaintext to a different and unique ciphertext. Thus, to decrypt the data, one must not only be in possession of the decryption algorithm, but also the secret key.

In designing security, one should assume Kirchhoff’s principle, “only the secrecy of the key provides security”. Specifically, the cryptographic algorithm2 is presumably already known to untrusted parties. Thus, only the cryptographic key must be kept secret.

Without knowing the key, an unbroken encryption scheme requires an attacker to do a brute force attack over the entire key space; for a given cipher text *c*, iterate through all keys in the key space, feed the decryption function with the given key and cipher text, and decide on the plausibility of the decoded plaintext *c′*. To make brute force attacks intractable, the key space should be extremely large.

3.3.1 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 [7] [1].

3.3.2 Public-Key Encryption (Asymmetric)

Boneh [14] proposed an encrypted searching technique based on public-key encryption. Using public key encryption addresses a weakness found in earlier proposals. In symmetric encryption, if Bob wishes to make it so that Alice can search an encrypted document, they must first agree on a secret key. The question becomes, how do they agree on a key without revealing it to others? However, in public key schemes, no such problem arises: he may use her public key. Thus, only Alice, who has the corresponding private key, may search the encrypted document[3].

3 In this naïve approach, only Alice can search the encrypted document; in less naïve approaches, by combining both public key encryption and symmetric key encryption, multi-user encrypted searching schemes can be constructed.

3.3.3 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 reconstruct the document from the information in the index. This particular relaxation offers a potentially more attractive option: Do not use a decipherable encryption scheme at all. Rather, use a one-way hash function [15] [16], ideally something that approximates a random oracle, and "filter" [17] the document (plaintext) through it, e.g., insert the one-way hash of each word (or however a searchable document term is defined) in the document into the secure index. In theory, it is impossible to tell which terms a document has by looking at the index (especially since the one-way hash is generally not invertible). Rather, one must transform words, or terms, using the same one-way hash construction, and check if there is a match on the transformation (it may be an erroneous match, of course). Note that this construction can be combined with one or more secret keys, as with encryption, to manage search authorization.

There are a couple of advantages to this approach. First, it is far less computationally demanding; executing a one-way hash is less computationally demanding than executing a decipherable encryption scheme. Second, because one-way hash functions are non-invertible and pre-image resistant[4], even if the secret keys are disclosed to unauthorized users, this does not necessarily compromise the contents of the actual document (except by allowing whatever search facilities the index permits).

4 TYPES OF INFORMATION LEAKS

Goh [13] contends that an encrypted search capability should reveal no information about the contents of the document unless a secret (called a trapdoor [14]—e.g., easy to compute, but difficult to invert) is known which allows the secret-holder to search it. 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 un-trusted party—like a cloud storage provider—captures an encrypted query, 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, subtle ways in which information can be disclosed—or "leaked". This remains true even if we assume an unbreakable encryption scheme is being used.

4.1 DOCUMENT CONFIDENTIALITY

Even if a strong cryptographic scheme is being used, information may still be leaked.

For example, consider the following. For each document in the collection, the words in a given document (and only its words) are passed through a one-way hash and that hash value is directly inserted into the index. Since this is a substation cipher for small units (words), it is vulnerable to well-known attacks. For instance, since it is likely the underlying word frequencies in the encrypted document collection are similar to other collections (corpus), statistical frequency analysis can be used to construct probable cipher string to plaintext word mappings. For reasons like this, Goh [13] argued traditional hash tables are not suitable for use as a secure index.

4 Given a hash value, it should be difficult to find an input for the hash function that outputs the given hash.

4.2 QUERY PRIVACY

The same argument for 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; it enables them to construct a model of an encrypted document by submitting many different plaintext queries to it and seeing which are relevant—e.g., in the case of Boolean keyword search, does it contain these keywords? Thus, it is vulnerable to basic dictionary attacks. Indeed, if the document supports phrase searching, single word keyword queries followed by 2-gram queries may be used to try to reconstruct a close replica of the original document.

A simple solution is to insert a cryptographic hash of the term, as elaborated on elsewhere[5]. However, this would not accomplish much, since it is not wise to assume the algorithm will be kept secret. If the algorithm is known, which is the operating assumption, the untrusted party may proceed the same as before. A better solution is to hash a term concatenated with a secret key. As long as the key is kept secret, only authorized users may submit queries to the secure index.

However, if a cryptographic query for a specific term always looks the same, then an untrusted party may slowly build up frequencies for cryptographic patterns in the indexes. If they combine this with prior knowledge, like the frequency distribution of English words, they may be able to figure out what each pattern is most likely representing (e.g., what assignment of terms to these cryptographic query terms maximizes the likelihood of some 1-gram language model).

A partial solution to this problem is to make it so that the same query terms will look different when being submitted to different encrypted documents. However, as elaborated elsewhere, this imposes serious challenges to the efficiency of relevancy (context-aware) scoring.

A much stronger (though somewhat impractical) solution is found in oblivious RAM [15].

4.3 ACCESS PATTERNS

To exacerbate matters, even if an encrypted search scheme provides robust data confidentiality and query privacy, access patterns and implicit information useful for statistical inference may still be leaked.

A user’s data access patterns constitute a certain kind of information. That is, what is the distribution of (encrypted) documents the user retrieved over time? For example, a particular user may show significant interest in a small subset of encrypted documents in the collection. If other

Another kind of information is revealed by a user’s query patterns. For example, if a private query is frequently followed by another action, like checking stock prices, this correlation may conceivably be used to infer properties about the query and the corresponding documents that are returned in response to it.

To mitigate these more subtle forms of information disclosure, Pinkas [15] proposed the use of oblivious RAM to conceal the history of patterns (observable by the server) associated with a user’s encrypted searching activities. Oblivious RAM may naively be thought of in the following way: to prevent meaningful statistics from being gathered about a user’s activities, whenever an action—a read or write—is performed, randomly include other randomly chosen actions (e.g., queries) to obscure what the user was actually interested.

5 See section 3.3.3 on one-way hash functions

4.4 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 private query.

Most encryption searching schemes [13] [1] [17] [18] implied this to be their primary objective, although only a few solutions, like that proposed in [15], considered maintaining this confidentiality standard when confronted with an untrusted party who considers the history of the user's activities.

5 ONLINE AND OFFLINE SEARCHING

Methods	**Advantages**	**Disadvantages**
Online [7], [10], [11], [1]	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 patterns; vulnerable to substitution cipher attacks Sequential search; slow and impractical for large-scale use
Offline: Inverted index [8], [2]	Extremely well-understood Context-aware searches are easier to perform Fast term lookups: O(log n) Space-efficient (if combined with Huffman compression) Data structure facilitates efficient ranking operations (see section 3)	Potentially vulnerable to substitution cipher attacks and other forms of cryptanalysis Potentially vulnerable to preimage attacks Leaks significantly more information than Bloom filter
Off-line: Bloom filter [13], [19]	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	*O(1)* is hiding a large coefficient; k cryptographic hash functions must be evaluated, where k can be large Deleting a term from a document requires re-construction of Bloom filter

5.1 ONLINE SEARCHING

Online search performs a sequential search on the document cipher [1] [5] [6] [9]. To be able to perform encrypted searches on such a cipher, a large block cipher may not be used; rather, each term must be encrypted separately to facilitate exact string matches on the encrypted query terms. In which case, this is a simple substitution cipher. As mentioned elsewhere, these may be broken, or at the very least leak information about the contents.

Moreover, as pointed out in [18], 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". If detected, the recipient can then be 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].

5.2 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; 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 [13] [19] like Bloom filters. A comprehensive overview of offline-based solutions can be found in [13].

5.2.1 Inverted index

In [2], a possible approach to a secure index is elaborated. Previously, this index was discussed in the context of using Huffman codes to serve as an efficient 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 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 analysis). Alternatively, an attacker could try various popular words to try mount a preimage attack on the values in the index.

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).

5.2.2 Bloom Filter (Signature File)

A Bloom filter [20] 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 (cryptographic) hash functions. For each member, use the k cryptographic hash functions to map it to k indexes in the bit vector, setting each bit to 1.

To verify an element is a member of the set, check to see if each of its k hash positions is set to one. On the one hand, If any are zero, then it is definitely not a member (false negatives are not possible). On the other hand, if all of them are set to one, we assume that it is a member. However, it is possible, with some false positive probability, that it is not a member; rather, one or more actual members caused those k bit positions to be set to one.

It is straightforward to construct a secure index from a Bloom filter. For each term (words, n-grams, or other searchable term 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 on the term concatenated with a secret, e.g., *insert(bloom, hash(term$_a$ | secret))*.

To 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 [13], 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, hash$_2$(hash$_1$(term$_a$ | secret) | doc_id)*. Likewise, during the construction of private queries, the user must apply the same transformations. Since *secret* is unknown to unauthorized parties, like the server, they are unable to query the secure index, and since the *doc_id* is different for each document, stronger guarantees on query privacy is provided.

However, do note this construction complicates search queries applied to an entire collection of documents. After all, the server itself cannot transform each of the query terms with the appropriate *doc_id*, the users must. If the users perform this task, then there will be considerable communications overhead since they will need to send a separate query for each document.

An important feature of Bloom filters, in the context of encrypted searching, is that they leak 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 a reason why Goh [13] argued the Bloom filter is such a strong choice. Additionally, the Bloom filter is 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 [20]) do not allow removal of members from the set, thus necessitating the Bloom filter's reconstruction whenever removal is required.

5.2.3 Problems with the Bloom filter

As promising as the Bloom filter is, it does not possess optimal space efficiency for a given false positive rate. The optimal space efficiency requires $-n \, log \frac{1}{\varepsilon}$, where ε is the false positive rate, but the Bloom filter is a factor of ~1.44 of that.

An example of an optimally space-efficient structure for a given false positive rate is a minimum perfect hash indexing into a vector with k bits per index (implying the the false positive rate is $\frac{1}{2^k}$).

Another problem with the Bloom filter is the fact that the $O(1)$ time complexity conceals a large coefficient. In a Bloom filter, the number of hash functions (that minimizes the false positive rate) is$-\frac{\ln p}{\ln 2}$, where p is the desired false probability level (this equation implicitly depends on m and n, where m is size of bit vector and n is cardinality of set). If p is 0.001—1 out of 1000 false positives—then k = 10. Furthermore, if conjunctive queries are used, in which it is problematic for any of the independent queries in the set to respond with a false positive, then the false positive rate for a conjunction of k queries is $p' = 1 - (1 - \mathrm{p})^k$. For example, if each independent query has a false positive rate of p = 0.001, then for a conjunction of 10 queries, the false positive rate is 0.01. A similar analysis can be done for disjunctions. Thus, even though a false positive rate of 0.001 may seem incredibly small, in practice it may not be sufficient depending on the supported query types.

6 HOW DO WE MAP QUERIES TO DOCUMENTS?

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

Methods	Advantages	Disadvantages
Boolean key-word search [7], [11], [1], [17], [21], [22] [23]	Simple Fast queries	Documents are either relevant or irrelevant (nothing in be-tween), so recall and precision suffer
Context-aware search [24], [25], [26], [27], [28], [18]	Much better precision and recall on results Draws from extensive research in the IR com-munity	Answering queries is more computationally demanding. (This may especially important in the context of cloud computing, where every second of CPU time is charged.) Causes more information leakage.

6.1 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. In addition, 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 indexed. For example, the solution proposed by Li [11] 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.

6.1.1 Extensions

6.1.1.1 Conjunctive keyword search

In [7], the authors 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 un-trusted 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 so-lution is still rather limited. First, they still only perform exact string matching (instead of approximate string matching). Second, 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. Finally, like most other solutions, they do not consider untrusted parties, which consider historical data [15].

6.1.1.2 Approximate keyword matching

In [12], 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.

6.1.1.2.1 Locality-sensitive hashing

In locality sensitive hashing, 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—locality-sensitive hash functions may be used as a form of dimensionality reduction [29]; 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 [30] 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, if this requirement is relaxed, then a choice of hash functions can be made which is 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.

6.1.1.2.1.1 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 may both be improved significant.

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 effective technique to improve precision and recall. [31]

6.1.1.2.1.2 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.

6.1.1.2.2 Edit distance
In [11], a mechanism is proposed to address the limitation in which only exact matches on keywords are performed. In particular, they propose a construction that allows for matches on typographical errors or typical spelling variations, e.g., “color” vs “colour”.

To accomplish this, when constructing the index, for each term in the document, add all size k edit distance 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.

6.1.1.2.3 Wildcard matching
Wildcard [32] 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 [11] can be readily repurposed to implement wildcard searching.

6.1.2 Exact phrase matching (word n-grams)
Most searchable encryption schemes only allow matches on keywords, but in [33], 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)

6.1.2.1 Biword [26] 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.

6.2 RELEVANCY (CONTEXT-AWARE)
As pointed out in [18], 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 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.

6.2.1 Types

6.2.1.1 Weighted keywords

Weighted keywords [27] are based on two fundamental insights. First, some of the terms occur more frequently in a document than other terms. 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:

$$term\_weight\_in\_document(t, d) = some\ function\ on\ the\ frequency\ of\ term\ t\ in\ d$$

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:

$$term\_weight\_in\_collection(t, D) = f\left(\frac{\|D\|}{\|\{t \in d, d \in D\}\|}\right)$$

Combining these two insights, we have tf-idf (term frequency, inverse document frequency) and its variants. An offline index, like a Bloom filter, may be adapted to work with these multiplicity metrics [34].

6.2.1.2 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—everything else being equals—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.

6.2.1.3 Semantic search

In [25], 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. 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 the meaning of the text is and whether the query corresponds to a similar meaning. This is a more complicated question. For instance, it may 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" and "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 accurately, e.g., "carnivore" maps to "carnivore-1" (word sense 1). Using an ontology (like *WorldNet*), 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 has already been 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.

7 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? [35] 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 $secret_1$ and $secret_2$
- User 2 is trusted to query secure index, partly, by sharing with him $secret_1$
- User 1 gives a server $secret_2$
- 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 $secret_1$ and $secret_2$ 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 $secret_2$ and he may not 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:

```
    $hash_2(hash_1(term_{plaintext} | secret_1), secret_2)$
```

Thus, to check the secure index for $term_{plaintext}$, both $secret_1$ and $secret_2$ 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 $secret_2$, then they will be able to continue querying the secure index.

8 REFERENCES

[1] 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.

[2] 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.

[3] G. G. Langdon, "Huffman codes," 2000.

[4] 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.

[5] C. T. a. D. L. Christian Collberg, "A Taxonomy of Obfuscating Transformations," 1997.

[6] D. Hofheinz, J. Malone-lee and M. Stam, "Obfuscation for cryptographic purposes," *TCC,* pp. 214-232., 2007.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] M. Celikik and H. Bast, "Efficient Fuzzy Search in Large Text Collections," *ACM Transactions on Information Systems,* vol. 31, no. 2, pp. 1-59, May 2013.

[13] E.-J. Goh, "Secure Indexes," *Trust, Privacy, and Security in Digital Business, Lecture Notes in Computer Science,* vol. 3592, pp. 128-140, 2005.

[14] W. Diffie and M. E. Hellman, "New directions in cryptography," 1976.

[15] B. P. a. T. Reinman, "Oblivious RAM revisited".

[16] C. Gentry, "Fully homomorphic encryption using ideal lattices," *Proc. STOC,* pp. 169-178, 2009.

[17] D. Boneh, G. D. Crescenzo, R. Ostrovsky and G. Persiano. , "Public-key encryption with keyword search," *Proceedings of Eurocrypt, Lecture Nodes in Computer Science,* May 2004.

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

[19] S. Artzi, C. Newport and D. Schultz, "Encrypted keyword search in a distributed storage system".

[20] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics,* vol. 1, no. 4, pp. 485-509, 2002.

[21] N. Cao, C. Wang, M. Li, K. Ren and W. Lou, "Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data," *Proceedings of IEEE INFOCOM,* April 2011.

[22] Y.-c. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," *Lecture Notes in Computer Science,* vol. 3531, pp. 442-455, 2005.

[23] 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.

[24] 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.

[25] F. Giunchiglia, U. Kharkevich and I. Zaihrayeu, "Concept Search: Semantics Enabled Syntactic Search," *Proceedings of CEUR Workshop,* 2008.

[26] R. A. Baeza-yates, "Text retrieval: Theory and practice," *In 12th IFIP World Computer Congress,* vol. I, pp. 465-476, 1992.

[27] C. Buckley and G. Salton, "Term-weighting approaches in automatic text retrieval," *INFORMATION PROCESSING AND MANAGEMENT,* vol. 24, pp. 513-523, 1988.

[28] H. Cao, D. H. Hu, D. Shen, D. Jiang, J.-T. Sun, E. Chen and Q. Yang, "Context-Aware Query Classification," *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 3-10, 2009.

[29] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," pp. 604-613, 1998.

[30] Y. Hua, B. Xiao, B. Veeravalli and D. Feng, "Locality-Sensitive Bloom Filter for Approximate Membership Query," *IEEE Transcations on Computers,* vol. 61, no. 6, pp. 817-830, 2012.

[31] R. Krovetz, "Viewing morphology as an inference process," pp. 191-202, 1993.

[32] R. Brinkman, P. Hartel, W. Jonker and C. Bösch, "Conjunctive Wildcard Search over Encrypted Data," *Proceedings of the VLDB International Conference on Secure Data Management,* pp. 114-127, 2011.

[33] Y. Tang, D. Gu, N. Ding and H. Lu, "Phrase Search over Encrypted Data with Symmetric Encryption Scheme," *2012 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW),* pp. 471-480, 2012.

[34] A. Kumar, J. J. Xu, J. Wang and L. Li, "Space-Code bloom filter for efficient traffic flow measurement," *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement,* pp. 167-172, 2003.

[35] Z. Kissel and J. Wang, "Verifiable Symmetric Searchable Encryption for Multiple Groups of Users," *Proceedings of SAM 2013,* 2013.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] S. Mehrotra, C. Li, B. Iyer and H. Hacigümüş, "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.

[41] B. Zhu, B. Zhu and K. Ren, "PEKSrand: Providing Predicate Privacy in Public-Key Encryption with Keyword Search," *Proceedings of IEEE International Conference on Communications,* pp. 1-6, 2011.

[42] R. Curtmola, J. Garay, S. Kamara and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," *Proceedings of the ACM Conference on Computer and Communications Security,* pp. 79-88, 2006.

[43] B. Xiang, D. Jiang, J. Pei, X. Sun, E. Chen and H. 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.

[44] Y. Shen and J. Yan, "Sparse Hidden-Dynamics Conditional Random Fields for User Intent Understanding," *Proceedings of the International Conference on World Wide Web,* no. Shuicheng, Lei Ji, Ning Liu, Zheng Chen, pp. 7-16, 2011.

[45] J. Chen, H. Guo, W. Wu and W. 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.

[46] M. Naor and M. Yung, "Universal One-Way hash functions and their cryptographic applications," pp. 33-43, 1989.

[47] R. Schenkel, A. Broschart, S. Hwang and M. Theobald, "Efficient Text Proximity Search," *Lecture Notes in Computer Science,* pp. 287-299, 2007.