# Encrypted Search

[Explain the growing trend, and the reasons why, more of our confidentially/private information is being stored on untrusted systems/servers—like cloud storage] [Explain how this requires encryption to provide data confidentiality, but that it is not reasonable to have to download the encrypted documents, then decrypt, then search since the document repository may be quite large or the capabilities of the client may be quite small.] [Explain how this calls for the need to be able to search the documents (on the server) without compromising data confidentiality—and also without comprising other information, like what you searched for, e.g., query confidentiality also.]

[But, before going into the details of encrypted search, explain the Big Picture of Information Retrieval in general.]

## Information Retrieval

Nearly everyone in the modern age is familiar with information retrieval (IR) in the form of Google. Indeed, Google has become a verb: “Let me Google that for you.” So, understandably, many might think this is the whole story. However, companies like Google have benefitted from decades of research by the IR community.

The term “information retrieval” was coined by computer scientist Calvin N. Mooers. In [?], he states IR is a method to convert users’ need for information into a list of references to documents containing information useful to them. This still essentially remains the central task of IR: finding any sort of **relevant** information to a specific **information need**, whether it be for email, music recommendations, or applications.

In other words, we have users who have a (characteristically vague) concept about what they are looking for, and we have a store of documents that express (characteristically ambiguous) concepts that are of interest to the user. The central challenge of IR, then, is to map the user’s need for information to a relevant subset of these stored documents.

This task, of mapping information needs to documents, is an incredibly deep and challenging problem. For instance, what problem are we even trying to solve? That is, what is the nature of relevance? [?] Relevance, as it pertains to humans, is a subjective measure: Is it on topic? Is it timely? Is it authoritative? In general, does it satisfy the user’s frequently ill-defined information need?

Or, as another example, when an IR system deals with unstructured natural language text, it must cope with an extremely ambiguous and richly expressive language. When a user enters the query, “saw the bird with a telescope”, to humans it is immediately clear what is meant, but syntactically the sentence is ambiguous; it is just as valid to interpret the sentence as meaning “saw a bird that was carrying a telescope”. Moreover, humans frequently disagree over the meaning of a sentence; natural language is inherently ambiguous, and properly must be dealt with probabilistically.

Furthermore, according to [?], the challenges in IR have only became more acute since the advent of the Internet enabling cheap and instantaneous access to an unprecedented and ever-increasing volume of information. When there is too much information—known as information overload—the benefit of having the information is outweighed by the burden of having to sort through it and understand it [?]. Thus, the issue of relevancy is of paramount importance.

A user neither wants to wade through too many results, whether relevant or irrelevant, nor waste their time paying attention to irrelevant results. The goal, then, is to optimize with respect to measures like precision and recall [?]. Respectively, minimizing the set of non-relevant[[1]](#footnote-1) documents returned while maximizing the set of relevant[[2]](#footnote-2) documents returned. These two measures are, in some cases, in direct opposition, as it is trivial to have absolute recall by returning every document in the collection, but this comes at the cost of degrading the precision to intolerable levels.

Thus, like many interesting problems in engineering, this a trade-off problem[[3]](#footnote-3), and the best answer usually depends on the context in which the question is asked.

## Implementing an IR system

Having established the importance of IR, especially in the modern age, the next logical question is, “How do we make one?” To make automated IR systems possible, the documents and information needs of users require symbolic representations which can be mechanically manipulated, ostensibly by a computer system. In general, the following decisions need to be made [?] in order to implement an IR system:

* What are the terms?
* How is term matching[[4]](#footnote-4) performed?
* What model will be used to facilitate relevancy scoring?
* What data structures will be used to facilitate fast and efficient retrieval?

In theory, its implementation can span the gamut from simple exact string matching to systems that can understand and use language superior to any human’s ability[[5]](#footnote-5).

### Syntactic searching

Roughly speaking, there are two (not mutually exclusive) approaches to IR: syntactic search and semantic search.

Syntactic search is, fundamentally, based on some measure of string similarity. That is, how similar are two strings? When one of the strings represents the information need of the user—in the form of a query string—implicitly the IR system is answering the question, “How relevant is this document to this query?”

How can syntactic searching be understood in the context of the four fundamental decisions mentioned previously? That is, what constitute the terms, how is matching performed, what model will be used, and what data structures will be used?

### Terms

One of the most important decisions to make in an IR system is deciding what the basic addressable unit of a documents is. These are called the terms of the document. There are two primary choices: single-word terms, and multi-word terms. The appropriate decision, in large part, rests upon the answer to a single question: should exact phrase matching be supported, as exemplified in Google by placing quotes around a phrase? This is an important question since exact phrase matching has proven quite useful to many people; indeed, as much as 10% of web searches include exact phrase searches. [?]

If exact phrase matching is desired, then to facilitate rapid searching, the terms may consist of both single-word units and multi-word units (n-grams). For example, the document “hello planet earth” may be represented as {“hello”, “planet”, “earth”, “hello planet”, “planet earth”, and “hello planet earth”}.

If false positives are tolerable, then a bigram model can be employed in which every consecutive pair of words are indexed. Thus, the search query for the exact phrase “hello planet earth” can automatically be transformed into “hello planet” and “planet earth”.

### Matching

The simplest kind of term match is equality—for a given query, does the document contain any of those exact query terms? The problem with this approach is that its recall suffers unnecessarily; it ignores typographical errors, common misspellings, and spelling and morphological variations. Matching on some subset of this is, in general, a form of (ranked) approximate matching[[6]](#footnote-6). Approximate matching can significantly improve an IR systems recall and, in some cases, precision.

#### Edit distance[[7]](#footnote-7)

A common approach to approximate matching tolerant to mishaps like typographical errors is edit distance. That is, apply up to k edit transformations on the query (or the terms of the query), where an edit is an insertion, deletion, or replacement of a single character. If the term can be transformed into a match in k transformations or less, then this may count as a possible approximate match.

When matching on such transformed queries, various distance measures have been proposed to measure how far a transformation is from the original query term, the most popular being the Levenshtein distance.

#### Locality-sensitive hashing

Locality sensitive hashing tries to hash “similar” (in some sense) inputs to the same output. This procedure is done mostly offline as a preprocessing step during the construction of the document representation (queries will also require preprocessing), the objective being to enable rapid approximate matching to improve the recall (and potentially precision) of a search result by matching inexact, but related, terms.

Simple examples—many may argue their inclusion as example of hashing—include ignoring case (converting every letter to lower-case), removing stop words (words that occur frequently in every document and so have little discriminatory power), and removing punctuation.

##### Stemming

A more sophisticated example is stemming[[8]](#footnote-8) (or relatedly, lemmatization) in which morphological variations of a word are mapped to a single base form. By reducing such variations to a single token, in which the different variations have the same essential meaning, recall and precision can hopefully both be improved.

Stemming is typically used to refer to a (linguistically inaccurate) heuristic in which the ends of words are chopped off—that is, stemmed. For example, the words “compute”, “computes”, “computed”, and “computing” may each be reduced to the stem “comput”. Despite their crudeness (“comput” is not even a word), stemming has demonstrated itself to be a fast and fairly effective technique to improve precision and recall. [?]

For instance, 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.

##### Phonetic algorithms

Phonetic algorithms map words that sound alike to the same hash. Soundex is one of the more popularly used examples of this; it is an especially useful trick for approximate matches on the names of people, which was its original purpose. [?]

#### Query expansion

Most of the previously mentioned ranked approximate matching techniques, like edit distance and stemming, can be considered under a more general category of query expansion, in which a query is conceptually transformed into multiple queries to improve the relevancy of the results. For instance, stemming can be seen as a query expansion technique in which all of the morphological forms of a query term are expanded, e.g., the query, “compute grades”, is conceptually expanded into the queries, “compute grade”, “compute grade”, “computed grade”, “computed grades”, …, “computing grades”.

At the risk of veering too much into semantic search territory, other forms of query expansion include finding relevant synonyms, e.g., searching for “computed grades” may be expanded to “calculated grades”, “determined grades”, and so on. This may be useful for the same reason stemming is useful: people use different words (and variations) to mean the same thing (this is known as the problem of synonymy); moreover, the problem can be exacerbated in the context of IR since documents often sample words from a different distribution than the less formal words people tend to use in queries to use to find relevant documents.

However, finding ways to effectively address this problem has proven to be a challenge. A general thesaurus transformation, in which synonyms are substituted for query words, often do little to improve the relevancy of results. [?] Other techniques, for instance automatically generating a localized thesaurus from the text (e.g., clustering based on word co-occurrence; whenever a word in a given cluster is used in a query, try out the other words in the cluster too) have demonstrated more promising results.

#### Wildcard queries

Wildcard queries may be thought of as a manual form of query expansion, in which a language is provided to expand the query in certain kinds of ways. Regular expressions are a robust example of wildcard queries.

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”. Often, techniques like stemming may already account for many such variations, but for more advanced and demanding users, the power to perform wildcard searches can be (at least perceived) as tremendously useful.

Wildcard queries in the context of IR tend to be far more limited than other types of queries, like regular expressions, due to concerns over speed and efficiency. IR systems typically create, as will be discussed later, a document representation which facilitates rapid search retrieval over a gigantic store of text. These representations limit the ways in which a string match may be efficiently found, e.g., using some of the more sophisticated string alignment algorithms for doing approximate substring matching may not be compatible with the index if it uses hashing to do look-ups in O(1) time.

So, in practice, wildcard queries often take more limited forms, e.g., an asterisks wildcard characters, in which the asterisks is used to represent a position which will match on any character. To efficiently support such wildcard queries, one can use relatively straightforward techniques, like the permuterm index structure.

### (Document) Models

The simplest model is to simply use the raw text. This is, certainly, the most flexible model—it is lossless, after all—but it does not facilitate rapid retrieval nor efficient relevancy scoring.

To efficiently and effectively retrieve relevant documents, the document is usually transformed into a representation that is more efficient for this task, e.g., facilitates fast term lookups while retaining sufficient information to do relevancy scoring.

#### Bag of words

The most common model in use is the **bag of words**(or terms) model. The only information represented in a document in this model is whether or not a term appears in the document, or some other measure, e.g., the number of occurrences of each term in the document. Thus, the document “Jack and Jill” is equivalent, in this representation, to the document “Jill and Jack”. While this is a rather lossy representation, it does appear to be reasonably effective. [?]

Due to this simplistic representation, this model makes the naïve but convenient assumption that each term in the bag of words is conditionally independent of every other term in the document. This is the same assumption made by Naïve Bayes classifiers. That is:

Where **relevant** is the class, either true or false, and **d0**, …, **dn** are the terms of the document, usually between 0 and 1.

Relevancy scoring metrics which are based on this underlying assumption made by the bag of words model will be covered in detail in this section, and only passing mention of scoring metrics which do not make this naïve assumption of conditional term independence will be made.

Boolean model [?]

The Boolean model returns the set of documents which exactly match the terms in the query. Therefore, it only provides a binary ranking of documents: either a document is relevant or it is not relevant. The query representation, however, tends to be fairly sophisticated, e.g., set-theoretic queries are possible, e.g., ***(“carnivore” or “dog” or “tiger”) and (“chase” or “hunt”) and (“cat” or “feline”)***. This kind of model is often employed by experts, although its usefulness has been disputed. [?]

The problems with this approach are numerous and

Heuristic model [?]

The heuristic model 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 brought out by the common-sense notion that a rarely seen term in a collection of documents is a more interesting event. Formally, according to Zipf’s empirical law, the frequency of any word in a corpus is inversely proportional to its rank in the frequency table, e.g., “the” is the most frequent word, and it appears twice as often as the second most frequent word, “of”, and three times as often as the third most frequent word, “and”.

In a store containing 1000 word types (where each word type may have multiple occurrences), then, the 25 most common words will be expected to comprise 50% of the words. Due to the lopsidedness of this distribution, the most common words will carry very little meaning; that is to say, they have no discriminatory power as they appear in nearly every document. Conversely, also due to Zipf’s lopsided distribution, many words will be very rare or even unique (hapax legomena). Indeed, hapax legomena typically account for around half of the word types in a corpus [?]. Thus, a large portion of the words will 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:

Where f increases as the input increases[[9]](#footnote-9). Combining these two insights, we wish to find some function g that combines these functions. A popular choice for these functions is called tf-idf [?], which stands for term-frequency, inverse-document-frequency. Mathematically:

Where f is, generally[[10]](#footnote-10), the raw frequency of the term t in document d.

A straightforward implementation that uses this heuristic is to simply sum the weights of each query term found in the document. However, other models, like the vector space model, can use the computed weights to inform their own brand of ranking. Aptly named, the heuristic method is a somewhat ad hoc approach but it performs comparatively well in practice.

Vector space model [?]

In the vector space model, documents are represented as vectors, where each term serves as an (orthogonal) dimension[[11]](#footnote-11) in a high-dimensional vector space, where the value along a certain dimension is 1 if the corresponding term is present in the document and 0 otherwise. A slightly more sophisticated approach is to use a weighting scheme, like tf-idf, to determine the document’s position in vector space.  
  
Thus, documents that have a similar distribution of words will be close to each other in this vector space. In turn, documents that are close together in this vector space will, it is hoped, be closely related to each other in some way.

Having constructed this representation of two documents, simple linear algebra operations, like the dot product or the angle between them (cosine similarity), may be used as similarity measure. Thus, the query string, when given such a vector representation, can then rank documents by this measure.

The vector space model can be readily applied to other algorithms, e.g., clustering, in which the result set for a query consists of documents assigned to the nearest cluster centroid.

Probabilistic model [?]

The probabilistic model is, in some sense, similar to vector space model, but relevancy is informed by probability theory. The argument goes, since relevancy is an uncertain science, this uncertainty should be explicitly represented.

As previously mentioned, this is essentially a Naïve Bayes classifier. Naïve Bayes classifiers have proven themselves to be surprisingly effective despite their seemingly over-simplifying assumption of conditional independence.

Language model [?]

A reasonable way to generate a query is to reason, “Which worlds would likely appear in the sort of documents I am looking for?”, and use those words to formulate one’s query. The language model implicitly models this approach to relevancy scoring.

The document is treated as a generative model, and the probability (or rather, likelihood) that a query would be generated (or sampled) by the document’s language model is used as the similarity measure. So, for instance, if a trigram language model of the document is used, then the probability that a query would be generated by the document’s language model is given by:  
Although more commonly, a unigram language model is used—which constitutes a **bag of words** (and Naïve Bayes) model.

#### PageRank and other link-based algorithms

…

#### Proximity’s effect on relevance

Another issue with relevancy not directly addressed in the previous models but pertinent to any practical scoring system employed is the issue of query term proximity. With respect to the query “hello world”, a document that contains both the terms is probably not relevant if they are separated by too many terms.

There are two primary approaches to deal with this. (1) Represent the documents at finer granularities, e.g., in the vector space model, every sequence of N sentences gets a different vector representation. 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. (2) Introduce a proximity measure, e.g., penalize matches in which the terms are too far apart. Both approaches tend to involve rather informal, ad hoc decisions, but it is necessary to make them to create an effective IR system.

### Data structure

There are several data structures that can be applied to the problem. The naïve data structure is the raw, unstructured text, or some transformation (e.g., stemming, lower-casing) thereof. In this case, a simple sequential search can be performed to service search queries. This has the benefit of not needing to be updated whenever the document is changed, and it being universally supported–sequential search can be found in nearly any document editor, and for good reason: it is highly useful. But, it has obvious limitations, the most obvious of which is slowness.

#### Inverted Indexes

For each term in a document, however a term is defined, insert it into a dictionary, where a dictionary refers to the positions of that word in the document. The position may be exact, e.g., byte offset, or it may be approximate, e.g., it is somewhere in block j (where block j consists of k terms).

Both approaches have strengths and weaknesses.

#### Signature Files

Signature files, like Bloom filters, may be used to provide a more probabilistic representation of documents. Bloom filters in particular are able to trade space complexity for the possibility of false positives on term matches. However, false negatives are not possible (in a classical Bloom filter).

More specifically, a Bloom filter is a probabilistic set membership data structure. It consists of an array of bit elements of some given size m, and a set of k independent hash functions that map all possible inputs to k (or fewer) integers within the range [0, m-1]. To insert an element into the filter, it is first mapped by each of the k hash functions, upon which each of the k positions is set to true without regard to collisions.

The more members that are inserted into the filter, the fewer 0’s there will be in the bit array. In turn, the fewer 0’s—the more 1’s—there are, the more likely an arbitrary input tested for membership will be mapped, by each of the k hash functions, to positions which have been set by other previously inserted members. In other words, the more members there are, the more likely a false positive will occur.

For this reason, Bloom filters are often only used as a quick and dirty check.

### Putting It All Together

Now that we have defined our terms, our matching criteria,

### Semantic search

However, problems of synonymy, polysemy, anaphora, …

#### Symbolic approaches

The meaning of the text can be explicitly represented, e.g., part of speech tagging or word-sense disambiguation. Once provided with an explicit representation, it becomes to perform more concept-based searching, e.g., “find me stuff that has a carnivore hunting prey”.

Natural Language Processing (Understanding/Generation)

* Tagging problem
  + Parts of speech
  + Named entity recognition
* Word-sense disambiguation
* Subsumption: general category -> more specific categories (carnivore-1 -> {dog-1, tiger-2})

#### Statistical approaches

Latent semantic indexing

1. Documents do not include the concept represented by the query, but worded sufficiently similar to. [↑](#footnote-ref-1)
2. Documents that include the concept represented by the query, but worded sufficiently different than. [↑](#footnote-ref-2)
3. Although frequently, both precision and recall can be improved in tandem. [↑](#footnote-ref-3)
4. Not to be confused with how relevancy is calculated—relevancy depends on matching terms, but matching terms does not depend on relevancy. [↑](#footnote-ref-4)
5. Beyond the current state of the art in natural language processing; it would probably require AGI. [↑](#footnote-ref-5)
6. Depending on the chosen data structure, different types of approximate matching may or may not be reasonable, e.g., a hash table representation of a document may not be able to efficiently support approximately matching on typographical errors. [↑](#footnote-ref-6)
7. [↑](#footnote-ref-7)
8. [↑](#footnote-ref-8)
9. Note that the input to the function f is the ratio of the number of documents in D to the number of documents in D which contain the term t, not the ratio of the sum of the terms in D to the frequency of term t in D. [↑](#footnote-ref-9)
10. To counter the bias a raw frequency metric would give longer documents, it can be normalized in some way, e.g., by its document length or by the frequency of the most frequent term in the document. [↑](#footnote-ref-10)
11. If term dependencies are modeled, then dimensions could represent something else, e.g., something informed by an n-gram language model instead of the unigram model implicit in the bag of words model. See generalized vector space model for more in-depth coverage. [↑](#footnote-ref-11)