Semantic Search Over Encrypted Data

Tarik Moataz*, Abdullatif Shikfa†, Nora Cuppens-Boulahia*, and Frédéric Cuppens*
*TELECOM Bretagne, Rennes, France
{tarik.moataz,nora.cuppens,frederic.cuppens}@telecom-bretagne.eu
†Bell Labs Research, Alcatel-Lucent, Nozay, France
abdullatif.shikfa@alcatel-lucent.com

***Abstract*—Searchable encryption is a promising primitive in the context of outsourced data storage and in particular cloud computing. Current constructions lack searching capabilities beyond exact search on keywords and are thus less efficient than plaintext algorithm that take into account the meaning and return results corresponding to semantically close keywords. We thus advocate for the need of semantic search over encrypted data. We study the state of the art in stemming algorithms and in searchable encryption and propose the first semantic symmetric searchable encryption constructions. This improvement over searchable encryption has similar security as the prior work of Curtmola et al. on symmetric searchable encryption.**

I. Introduction

With the advent of cloud computing, outsourcing storage has become an increasingly popular trend and many users are storing their data in the cloud in order to benefit from *unlimited* storage space. Some of this data is sensitive and must be protected from unauthorized access, including from cloud operators, classically considered honest-but-curious. A classical solution in this setting is for users to encrypt their data before sending them to the storage server. This solution protects the data but is at odds with the utility of the cloud, as the data becomes completely obfuscated and the cloud operator cannot extract any information from the data nor perform searches on it.

One interesting approach to solve this dilemma and allow searches on encrypted data lies in searchable encryption. This recent concept was tackled in the early 2000s by Song et al. in [19] where the aim is to enable a user to store his encrypted data in a remote server, then sending encrypted search queries and retrieving only the encrypted documents matching the search. The entire search operation is done on the server side without leaking any information concerning either the content of the query or the content of the documents. Searchable encryption has attracted a lot of attention since that first work, however, to the best of our knowledge, there is no scheme in the literature that tackled the issue of semantic search over encrypted data. In plaintext information retrieval, semantic search is enabled thanks to a variety of techniques including for example stemming algorithms which improve the efficiency of the search. In few words, such techniques enable a user to search not only documents containing a given keyword, but all documents containing related keywords which have a close meaning. It is natural to try to extend this kind of features to encrypted searches as well and our problem can then be formulated as follows: how can we enable semantic search over encrypted data, that is to say encrypted searches that enable to retrieve documents containing all keywords meaningfully related to the keyword sent in the query.

A. Related Work

As we mentioned, our work falls within the framework of searchable encryption. After the initial work of Song et al., several improvements have been proposed both in the symmetric setting [9], [7], [5] and the asymmetric one [3], [1], [2]. However all these schemes only enabled exact encrypted searches consisting of a single keyword.

Starting from the year 2004, other works (e.g. [10], [4], [20], [18]) tackled the issue of improving search options over encrypted data. These works have focused mainly either on conjunctive exact keywords search or conjunctive subset search or conjunctive range query search.

Finally another closely-related field one could think of to solve our problem is "fuzzy private matching". Fuzzy private matching aims at securely detecting a match between strings belonging to different parties even if the two strings are similar but not equal, in the sense that t out T letters are at the same places. Freedman et al. presented a first solution [8] of 2 out 3 letters, however their solutions presented some flaws described in [6] who presented a solution for any value of t and T where $t < T$. Many works appeared later to reduce the protocol communication complexity in order to avoid incurring a factor of $\binom{t}{T}$ [22]. However, fuzzy matching is useful to remove the effect of errors or for specific applications like comparison of replicated DNAs (where some mutations might randomly occur) but it does not take into account the meaning of strings at all: two words with a short lexicographic distance are not necessarily close from a semantic perspective, and on the contrary two words with a relatively large lexicographic distance might be close in the semantic plane.

B. Our Contribution

In this article we present an original solution that enables semantic search over encrypted data. To achieve this goal, we first studied the state of the art techniques to perform semantic search over plaintext data, as such techniques enhance the performance and more generally the efficiency of information retrieval mechanisms. Most of these techniques are based on a variety of stemming algorithms. This technique has shown its efficiency for plaintext data, thus we aim to use it in an encrypted context.

From this point, our contribution is to combine an efficient searchable encryption technique with a stemming algorithm in order to perform semantic search over encrypted data. In the basic searchable encryption scheme one keyword is associated to all its corresponding documents. In order to allow semantic search, we have to associate all related keywords to the same set of documents. Related keywords have the same root and consequently we can only store the root of related keywords in the secure index. In the search phase, the user enters the keyword that he wants to search for, then automatically, the stemming algorithm extracts the root of this keyword and finally the root is sent instead of the keyword in the query. As a result, the user retrieves all encrypted documents which contain all related keywords. By mixing one of the most efficient searchable encryption schemes and a stemming algorithm we can thus achieve semantic searches over encrypted data.

C. Outline

The rest of the article is as follows: in section 2 we introduce an overview of the existing stemming algorithms while section 3 recalls Curtmola et al.'s scheme by briefly explaining its encryption and search phases. In section 4 we present the details of our original construction of semantic search over encrypted data. Finally we conclude in section 5.

II. Stemming Algorithms

In this section we present stemming algorithms which are fundamental building blocks of most semantic search mechanisms. For this purpose we present on the one hand an overview of stemming algorithms and depict features that they achieve, and on the other hand we give a brief but exhaustive classification of these algorithms.

A. Stemming Algorithms Overview

Stemming is a general morphological process analyzer which aims first at identifying words having roughly the same meaning, in particular words which have the same root, and associates this set of words with the root (called also stem if no prefixes were removed from this set of words during the process). We should point out that despite the fact that all stemming algorithms rely on morphological language aspects, the purpose of stemming algorithm is not to find the correct meaningful root of words. Indeed, the root might very well have an incorrect form with respect to the lexical rules, for example the word "ease" can have "eas" as a root result. On the contrary, the process aiming at normalizing words and finding the correct morphological stem is called the lemmatization: the latter proceeds with algorithms which find the canonical form of lexemes (lexemes represent keywords which have the same root). In semantic search algorithms, the query stemming is performed obliviously for the user, thus the morphological incorrectness of word's stems is not a problem, and having "ease" or eas" will not change anything for the query results. Another point to emphasize is that basic stemming algorithms do not take into account the grammatical or lexical relationship, instead they take assume that words having the same root are semantically close.

B. Stemming Algorithms Classification

The literature is abundant of stemming algorithms, which take advantage of different techniques such as suffix-stripping algorithms, n-gram algorithms, stochastic algorithms and so on. These algorithms can be classified in three main types: affix stripping, statistical stemming and mixed stemming.

1) Affix Stripping: Affix stripping algorithms apply procedurally several rules in order to remove known suffixes and prefixes from words in order to identify the "root". As leading stripping stemmer examples we can quote the first removing stemmer algorithm introduced by J.B. Lovins [11] in 1968 and the well known and most used Porter suffix stemmer algorithm introduced in 1980 [17] and developed into a widespread framework called "Snowball". However this type of algorithm has the disadvantage of imperatively requiring an *a priori* knowledge of the language to build rules of affixes removing. In the following we enter into more details of the Porter algorithm for a better understanding of the affix stripping algorithms.

Porter algorithm: Porter algorithm aims at removing suffixes from words in order to find the root. We can imagine the following set of terms : CONNECT, CONNECTED, CONNECTING, CONNECTION, CONNECTIONS (example taken from [17]); these words are lexemes which have similar meaning, and the stem of these lexemes is CONNECT. The operation of finding the stem is done by automatic means: the algorithm defines a set of rules (about 60 but they can be extended further) and words are then confronted to all these rules sequentially before outputting the stem. The rules have the following form: (condition)(old suffix)⇒(new suffix). Here are for instance two examples of rules:

if the word length > 0 *and ends in 'ing', remove the 'ing'.*
if the word has at least one consonant and one vowel and ends in 'eed', replace the 'eed' by 'ee'.

The latter means for example that "agreed" becomes "agree" while "feed" remains unchanged. In summary Porter's algorithm is quite simple and its wide spread implementation "Snowball" reflects its popularity. The algorithm is not generic in that one needs to define different rules for different languages, but once these rules are defined for a language the algorithm performs well and can have many applications.

2) Statistical Stemming: As explained in the previous subsection, the knowledge of the language used is the major disadvantage in stemmer algorithms. To overcome this limitation, Mayfield and McNamee introduced the n-gram stemming algorithm [13] which does not require the definition of the language as input. Indeed their idea consists on doing statistical tests on n-grams (recall that an n-gram is a group of n consecutive letters in a word) existing in the document (whatever the document language): the n-grams which have the bigger frequency are considered as affixes; on the contrary the n-grams which have the lowest frequency are classified as

roots. This algorithm has good results for stemming without having the knowledge of the language and the authors demonstrated its efficiency by testing it for eight languages. There exists other interesting statistical stemmers besides Mayfield and McNamee's n-gram method such as the one introduced by Melucci and Orio based on the Hidden Markov Models [14] and the one introduced by Majumder et al. [12] based on clustering words and defining stems as the centroids of each cluster. As the name of this category indicates it, all these stemmers have in common to be based solely on statistics on words or subwords in a text, their performance is good when the language is unknown, but comparatively their performance is lower than affix stripping algorithms if the language is know.

3) Mixed Stemming: Mixed algorithms aim at overcoming the limitations of the previous approaches namely the context sensitivity and the corpus related features. Indeed, affix stemmers tend to conflate words with similar syntaxes but different semantics. This difference between words having the same root is due to the corpus based issue. On the other hand, we can find several inflected words used in different documents which deal with different contexts and have the same syntactic root, however their semantic meaning is different, an issue that we can tackle by using statistical stemming. Consequently the user ends up retrieving documents which are not semantically related to its query. The first proposed solution [**?**] to deal with this issue consisted on adding look-up tables to affix algorithms in order to store some linguistic exceptions such as the irregular verbs (buy⇒bought) and the irregular plurals (foot⇒feet). However this solution decreases the performance of the algorithm and does not solve some contextual issues such as the fact that "news" and"new" are both stemmed to the same root "new" but their meaning is quite different. Few advanced schemes were proposed to deal with this issue by taking into account the context of words in documents, and we can in particular cite the corpus-based approach [21] by Xu and Bruce and the more recent development described by Peng et al. in [16].

This concludes the overview and classification of stemming algorithms which are used in particular to perform semantic searches over plaintext data. In the next section we shift our focus to the most efficient searchable encryption mechanism.

III. Searchable Symmetric Encryption

We already mentioned in section I-A that there are several searchable encryption schemes in the literature. In this section, we provide some details on the encryption and search phases of Curtmola et al.'s approach called Searchable Symmetric Encryption SSE [7] as it is the most efficient scheme in terms of search complexity and we will use it as one of the building block of our construction.

a) Notations: Let $\mathcal{D} = \{D_1, ..., Dn\}$ be a collection of n documents, and $\Delta = \{W_1, ..., W_l\}$ be the set of unique keywords over $\mathcal{D}$. We also denote by $\mathcal{ID}(W_i) = \{id_{i,1}, ..., id_{i,m}\}$ the set of identifiers of documents that contain the keyword W_i, and by $\mathcal{ID}(\mathcal{D}) = \{\mathcal{ID}(W_1), ..., \mathcal{ID}(W_l)\}$ the collection of all keyword identifier sets.

A. Encryption Phase

The output of this phase is the collection of encrypted documents and a corresponding index $(\mathcal{E}(\mathcal{D}), \mathcal{I})$ where $\mathcal{E}$ represents a symmetric encryption scheme such as AES. The index $\mathcal{I}$ is constructed as follows. The identifiers sets $\mathcal{ID}(W_i)$ are represented in lists L_i which are stored in an array $\mathcal{A}$:

$$\mathcal{A} = \{L_1, ..., L_l\}.$$

$L_{i,j} =< id_{i,j}, pt_{i,j} >$ then represents the j^{th} element of L_i and it is composed of the identifier $id_{i,j}$ of the j^{th} document containing W_i and of the memory address $pt_{i,j}$ of the next element in $\mathcal{A}$. To sum up, the linked List L_i has then the following structure:

$$L_i = \{< id_{i,1}, pt_{i,2} >, ..., < id_{i,m-1}, pt_{i,m} >, < id_{i,m}, NULL >\}$$

The array $\mathcal{A}$ is then scrambled uniformly at random such that the next position in the array does not necessarily correspond to the next element in the linked list (hence the importance of $pt_{i,j}$), as illustrated in figure Fig. 1:

$L_{i,1}$	$L_{j,l}$	$\cdots$	$\cdots$
$\cdots$	$L_{j,2}$	$\cdots$	$L_{i,5}$
$\cdots$	$L_{j,1}$	$\cdots$	$\cdots$

Fig. 1. The Scrambled array $\mathcal{A}$

At this stage, for each keyword W_i, m_i keys are generated such that $\mathcal{K}(W_i) = \{k_{i,0}, k_{i,0}, ..., k_{i,m_i-1}\}$ and in each linked list L_i, the pointers are encrypted as follows:

$$L_{i,j} =< id_{i,j}, \mathcal{E}_{k_{i,j-1}}(pt_{i,j+1} \| k_{i,j}) > .$$

Finally, a look-up table $\mathcal{T}$ is required to provide the point at the first element of each linked list. $\mathcal{T}$ consists of couples containing the encryption of a keyword W_i under a secret key K_1 and the concatenation of the pointer $pt_{i,1}$ to $id_{i,1}$ and a key $k_{i,0}$ xored with a hash function H_{K_2} of the keyword:

$$\mathcal{T}_i =< \mathcal{E}_{K_1}(W_i), (pt_{i,1} \| k_{i,0}) \oplus H_{K_2}(W_i) > .$$

Note that $\mathcal{E}_{K_1}(W_i)$ represents a virtual address recognized in a FKS dictionary[15], thereby the time for checking $pt_{i,1} \| k_{i,0}$ is constant.

At the end of this phase, the user sends the index $\mathcal{I} = (\mathcal{E}(\mathcal{D}), \mathcal{A}, \mathcal{T})$ to the server.

B. Search phase

To search for a word W_i, the user sends the following trapdoor to the server:

$$(\mathcal{E}_{K_1}(W_i), H_{K_2}(W_i)).$$

The computation performed at the server side are efficiently summarized in figure Fig. 2.

At the end of the search phase, the server sends the list $\mathcal{ID}(W_i)$ of all the identifiers of the word W_i.

IV. Semantic Symmetric Searchable Encryption

As stated in the introduction, our aim is to enable semantic search over encrypted data. Our approach is to combine stemming algorithms with searchable encryption ones.

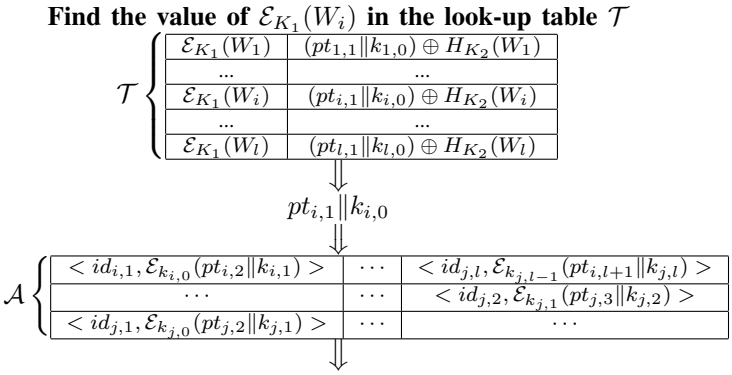

Fig. 2. Search phase process

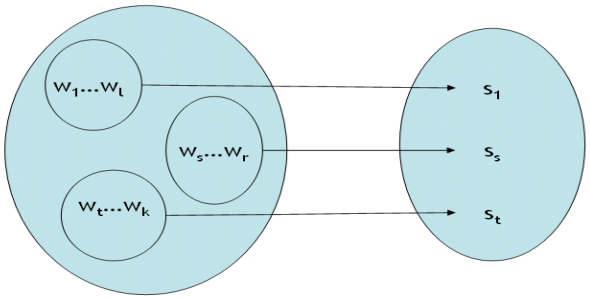

Fig. 3. Stemming process

A. Constructions

1) Initial Approach: The first idea is to simply combine stemming and searchable encryption sequentially without modifying any of them. Indeed, in order to retrieve documents containing keywords having the same meaning as a given keyword, the user could generate from this original keyword the set all the keywords within proximity in a semantic sense. For example, from the keyword "search", he generates "searching", "searches", "research", etc. Then the user has to send as many queries as the number of semantically close keywords to the server, which answers with the documents corresponding to each keyword. This basic solution satisfies basic requirements of semantic searchable encryption however it has several shortcomings: it requires multiple communication rounds and several search computation in the server side (the increase is proportional to the number of semantically close keywords). Furthermore, several semantically close keywords could belong to the same document which will then be sent several times, which is not optimal. Finally this approach might also be less secure as the server learns that several semantically close keywords are being searched, which is an additional information leakage which impact is not well understood yet. All these drawbacks lead us to consider the following other approach.

2) Improved Construction: Our second construction is based on a modification of Curtmola et al.'s scheme [7] integrating a stemming algorithm. First of all, we stress that our construction works with any stemming algorithm, however, the choice of the stemming algorithm will implicitly impact the strategy of information retrieval. As shown before, there are several types of stemming algorithms, each one has some features which make it more suited for semantic or syntactic retrieval, and the user has to chose the one which is more appropriate for his strategy. In our first implementations we have decide to use Porter's algorithm [17]. Our construction aims at associating documents with the root of all the related lexemes. This association extends the scope of the encrypted documents retrieval to all keywords sharing the same stem and consequently with nearly the same meaning.

a) Settings: In addition to the notations introduced in section III, we denote by S a stemming algorithm which takes as input the set of all keywords $\mathcal{W}$ and outputs the stems of all these keywords denoted by $\mathcal{S} = \{s_1, ..., s_r\}$ such that $|\mathcal{S}| < |\mathcal{W}|$. As shown in figure Fig. 3, each stem s_i in $\mathcal{S}$ is associated to a unique sub-set of keywords denoted by $\mathcal{W}(s_i) \subset \mathcal{W}$. Moreover, since each keyword has one and only one stem, we then have

$$\forall i, j \in \{1, ..., m\} \ s.t. \ i \neq j, \ \mathcal{W}(s_i) \cap \mathcal{W}(s_j) = \emptyset.$$

We denote by $\mathcal{ID}(s_i)$ the set of all document identifiers containing at least one element within the set $\mathcal{W}(s_i)$:

$$\mathcal{ID}(s_i) = \bigcup_{w_j \in \mathcal{W}(s_i)} \mathcal{ID}(w_j).$$

b) Encryption Phase: The encryption of the document is the same as in [7]. Concerning the construction of the index we add a preliminary step performed by the user, which consists on extracting the stems of the keywords: using the stemming algorithm S, we generate from the set $\mathcal{W}$, the stem set $\mathcal{S} = \{s_1, s_2, ..., s_p\}$. For each s_i in $\mathcal{S}$ we also generate $\mathcal{ID}(s_i)$ as explained in the previous paragraph. The rest of the construction is similar to Curtmola et al.'s scheme, with the exception of the substitution of keywords by stems. The look-up table contains the encryption of the stems instead of the encryption of the keywords, thus the i^{th} entry in $\mathcal{T}$ is $< \mathcal{E}_{K_1}(s_i), (pt_{i,1}\|k_{i,0}) \oplus H_{K_2}(s_i) >$, and the scrambled array the document identifiers of stems in $\mathcal{ID}(s_i)$ instead of document identifiers of keywords in $\mathcal{ID}(W_i)$.

c) Search Phase: In order to search for documents containing not only a given keyword W_i but also all the documents containing semantically close keywords, the query for W_i is replaced by a query for the stem s_j such that $W_i \in \mathcal{W}(s_j)$. This stem is extracted automatically from the stemming algorithm S. The query sent for W_i is in this case $(\mathcal{E}_{K_1}(s_j), H_{K_2}(s_j))$. We should emphasize that the modification is only at the client side: once the server receives the query, the search phase resumes similarly to what was

presented in section III-B, as the server already has indexes for the stems and not the keywords.

B. Analysis

The improved construction enables the user to search for all documents containing the word W_i and also all the lexemes that are similar to W_i. Similarity is underlined by the fact that the lexemes and the keyword W_i have the same stem. As mentioned previously retrieving all these documents will enhance the efficiency of the information retrieval for the user as it returns documents which are relevant even though they do not contain the exact keyword. From a functionality perspective, the construction is consistent and achieves the basic features of semantic symmetric searchable encryption.

1) Security Analysis: From a security perspective, semantic symmetric searchable encryption is secure in the same sense as Curtmola et al.'s scheme and in the same models defined in [7]. The additional stemming phase is indeed performed locally at client side and does not affect the security of our construction. However the access pattern defined by the association between the query and the resulting documents is modified and queries return more documents on average in our construction compared to [7]. This increase helps a malicious attacker in learning more relations between the documents but at the same time the growth creates noise to any adversary who wants to extract information from the access pattern. To completely get rid of this issue is it possible to use the second scheme introduced in [7]. The latter enhances the achieved security level as it is secure in a more stringent model (which takes into account an adaptive adversary) but it slightly increases the storage overhead in the server side. We have not described this solution in this article for the sake of clarity, but it is compatible with our approach.

2) Computation Complexity: From a computation perspective, the search phase consists in searching over stems stored in encrypted form in indexes instead of unique keywords in documents. Since the stem set is smaller than the keyword set $\mathcal{S} \leq W$, this means that in practice this approach has a lower storage and search overhead at the server side: searching for a stem is computationally less expensive than searching for keywords. However, the construction adds a stemming phase which increases the computation on client side. However this step can be performed in an offline only once by the user. Furthermore, the bottleneck in computation is considered to be at the server side as the server has to answer the queries of many clients, hence the overhead at client side is advantageously compensated by the gain at server side.

V. Conclusion

Searchable encryption is an interesting paradigm with great potential however its utility is reduced by the fact that previous works focused on exact keyword search, hence its performance was reduced compared to plaintext search. After having identified this issue and defined the need for semantic searches over encrypted data, we analyzed the state of the art in plaintext search, and in particular stemming algorithms. We then proposed the first constructions that achieve semantic symmetric searchable encryption.

These constructions are initial contributions and we plan to improve them on the following aspects. First of all the current constructions consider all lexemes in a given stem set uniformly and in particular do not distinguish between exact search and semantically close keywords, while the former has clearly more value than the latter. More generally we want to improve the relevance of the search by introducing different weights depending on the semantic distance between keywords. The search relevance would also benefit from searches on encrypted data supporting the conjunction of several lexemes.

References

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptology*, 21(3):350–391, 2008.

[2] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. *CRYPTO*, pages 535–552, 2007.

[3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. *EUROCRYPT*, pages 506–522, 2004.

[4] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. *TCC*, pages 535–554, 2007.

[5] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. *ACNS*, pages 442–455, 2005.

[6] L. Chmielewski and J.-H. Hoepman. Fuzzy private matching (extended abstract). pages 327–334, 2008.

[7] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[8] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. pages 1–19, 2004.

[9] E.-J. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[10] P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. *ACNS*, pages 31–45, 2004.

[11] J. B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31, 1968.

[12] P. Majumder, M. Mitra, S. K. Parui, G. Kole, P. Mitra, and K. Datta. Yass: Yet another suffix stripper. *ACM Trans. Inf. Syst.*, 25(4), 2007.

[13] J. Mayfield and P. McNamee. Single n-gram stemming. pages 415–416, 2003.

[14] M. Melucci and N. Orio. A novel method for stemmer generation based on hidden markov models. pages 131–138, 2003.

[15] L. F. Michael, K. János, and S. Endre. Storing a sparse table with 0(1) worst case access time. *J. ACM, 31(3)*, 1984.

[16] F. Peng, N. Ahmed, X. Li, and Y. Lu. Context sensitive stemming for web search. pages 639–646, 2007.

[17] S. E. Robertson, C. J. van Rijsbergen, and M. F. Porter. Probabilistic models of indexing and searching. pages 35–56, 1980.

[18] E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. *IEEE Symposium on Security and Privacy*, pages 350–364, 2007.

[19] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[20] P. Wang, H. Wang, and J. Pieprzyk. Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups. *CANS*, pages 178–195, 2008.

[21] J. Xu and W. B. Croft. Corpus-based stemming using cooccurrence of word cariants. *ACM Trans. Inf. Syst.*, 16(1):61–81, 1998.

[22] Q. Ye, R. Steinfeld, J. Pieprzyk, and H. Wang. Efficient fuzzy matching and intersection on private datasets. pages 211–228, 2009.