Usable Secure Private Search

Mariana Raykova, Ang Cui, Binh Vo, Bin Liu,
Tal Malkin, Steven M. Bellovin and Salvatore J. Stolfo
Department of Computer Science
Columbia University
New York NY, 10027, USA
{mariana,ang,bl2329,binh,tal,smb,sal}@cs.columbia.edu

November 6, 2011

Abstract

It is a common requirement in real world applications for untrusting parties to be able to share sensitive information securely. We describe a secure anonymous database search scheme (SADS) that provides exact *keyword* match capability. Using a new primitive,*re-routable encryption*, and the ideas of Bloom filters [1] and deterministic encryption [7], SADS allows multiple parties to efficiently execute exact match queries over distributed encrypted database in a controlled manner. We further consider a more general search setting allowing similarity searches, going beyond existing work that considers similarity in terms of error-tolerance and Hamming distance [8,11] by capturing semantic level similarity in our definition. Building on the cryptographic and privacy preserving guarantees of the SADS primitive, we then describe a general framework for engineering usable private secure search systems.

Keywords: *Database search, query processing, privacy, storage, retrieval, sorting, feature extraction*

1 Introduction

The ability to securely share sensitive information between untrusting parties is a prerequisite for many real-world applications. For example, consider a hypothetical criminal investigation database. For obvious reasons, access to details pertaining to ongoing investigations should be strictly controlled. On the other hand, we may wish to accommodate investigators who wish to determine if other investiagtions are related to their own, by allowing them to query the database. In doing so, we would need to anonymize the identity of the querier and content of his queries to ensure no details of his investigations are compromised. Furthermore, the database must truthfully execute all queries, returning any and all relevant information based on the query given without revealing information about unrelated investiagtions.

Before tackling the challenges of flexibility and usability of such a database, we first describe one secure anonymous database search scheme, called SADS, that provides exact *keyword* match capability. By using a new primitive,*re-routable encryption*, along with Bloom filters [1] and deterministic encryption [7], SADS allows multiple parties to efficiently execute exact match queries over distributed encrypted database in a controlled manner. Furthermore, it addresses the more complicated problem of allowing document similarity searches. While there is a pool of existing work considering similarity in terms of error-tolerance and Hamming distance [8,11], we capture semantic level similarity in our definition, as well as show how to apply this work in an efficient encrypted search scenario, something which has not been done before.

There are preexisting schemes for encrypted search that provide these privacy guarantees, but at a provably high efficiency cost. As such, they scale poorly, and are not suitable for real world applications with very large databases. The SADS system makes use of additional third parties and relaxed definitions of security to circumvent these inherent efficiency costs. While SADS provides one desirable set of security and efficiency guarantees, it lacks flexibility and semantic awareness of the corpora which it operates over. We extend SADS in two ways: we identify varying security and efficiency needs of different settings and provide a modular framework for adapting the system to meet them, and we expand its search capabilities beyond exact keyword match.

There also exist several systems in the commercial sector that provide encrypted storage in a cloud setting, but unlike our system, those that provide searchability do so under the querier-owned model: the owner of the data is the one querying it. Mozy and Carbonite are online backup services which provide on-disk encryption of files, however this encryption is not searchable. [14, 18]. Dropbox and Cloud Experience are cloud services that further provide file replication, synchronization, sharing and backup services and encrypt data on disk, also without search. [15, 16] Sugarsync and Box.net provide similar services while encrypting data during transit, but still do not provide search functionality. [13, 21] Evernote is another cloud service intended to provide widespread availability and searchability of user-written notes. Text-only portions of these notes may be encrypted, but those portions which are encrypted are no longer available for search. [17] Vaultive and Navajo Systems (recently acquired by SalesForce) provide access to encrypted data through a proxy, optionally allowing searchability via deterministic encryption of individual tokens. [19, 22] These companies assume the user issuing the queries is querying his own data stored encrypted remotely.

Using SADS as the foundational building block, we describe a framework capable of creating flexible query systems which still deliver strong cryptographic and privacy preserving guarantees. Furthermore, we

demonstrate how such a framework can be parameterized to adapt to a spectrum of security and usability requirements. The result is a modular system with a well-defined set of compile-time switches that allow the system designer to select the appropriate combination of the strictness of matching behavior, flexibility of use and computational overhead for a specific document corpus without changing the fundamental organization of the system.

Further, the exact match constraint imposed by SADS, and Bloom filter based querying in general, does not allow for search capabilities like case and grammatical number insensitivity. To overcome this limitation, we use the notion of context-specific, semantically aware *feature extraction*, and apply it to encrypted search scenarios. Using this abstraction mechanism to preprocess both input data and queries, we demonstrate ways to significantly augment the search capabilities of all Bloom filter based querying systems. As Section 2 shows, effective feature extraction is pivotal to both the integrity and usability of the private secure search system.

The remainder of the paper is organized into four sections. First, Section 3 defines the security and privacy requirements of our search system and details SADS, a secure private keyword search scheme based on re-routable encryption and Bloom filters. Section 2 anecdotally illustrates several important complexities of engineering private secure search systems over real-world corpora and demonstrates the necessity of context specific, semantically aware pre-processing prior to using SADS. Section 4 builds on the previous two sections and formally presents a framework for engineering usable secure private search systems by combining the strong guarantees of SADS with the flexibility of semantically aware document feature extraction and parameterizable query matching behavior. Section 5 discusses a real-world implementation of secure private database capable of executing flexible queries over RFC documents as well as Bible verses. Lastly, concluding remarks and directions for future research are presented in Section 6.

2 Motivation: Flexible Queries in an Exact Matching World

Creating accurate, user-friendly query systems which still exhibit strong cryptographic and privacy guarantees presents several challenges when we consider the gamut of documents which we will encounter. Furthermore, the context in which such systems are used significantly impacts the desired behavior of the query system, possibly altering the definition of terms like false positive and false negative. In short, we would like to create a private secure search framework that is *flexible*, in order to achieve reasonable usability, *parameterizable*, in order to adapt to diverse environments and privacy requirements, *multilingual*, in order to incorporate documents of different languages, medias and formats, while at the same time provide strong cryptographic and privacy guarantees described in the previous section.

The fundamental conflict between the exact matching search primitives and the overall functional requirement of flexible, fuzzy search system is easy to see. The remainder of this section illustrates common challenges of applying exact match keyword search to structured documents containing natural language content. In each of the problems presented below, we propose specific augmentations to the query system to improve usability and flexibility. Lastly, we draw several general observations about the proposed augmentations, culminating in the general private secure search framework, which is formally presented in Section 4.

2.0.1 Search Over Simple Words

Consider an example data set containing the following simple words:

Cat, dog, Bird, Lion, frog, Ang, Mariana

Conventions of natural language can complicate even a simple query system like the one shown above. A direct application of secure keyword search requires *exact* lexicographical match to yield a positive search result. Consequently, slight variations in capitalization and punctuation between the query and queried terms

will yield negative search results. For example, queries like *"birds"* or *"Mariana's"* will yield no matches unless the exact lexicographical match system is augmented to account

Depending on the context in which this query system is deployed, such behavior may be desirable. However, awareness of language conventions and a controllable degree of matching flexibility can greatly enhance usability.

In order to do this, we can *preprocess* the input during insertion and query operations. As we show in Section 4, the careful construction of preprocessing *feature extractors* play a pivotal role in defining and controlling the level of flexibility a secure query system will tolerate. Feature extractors are context sensitive, and should be tailored to the specific document corpora and the desired level of matching flexibility which query system is expected to operate.
Consider the following sets of 1-grams.

```
Feature Extractor A
  [cat, dog, bird, lion, frog, ang, mariana]
Feature Extractor B
  [cat, dog, bird, lion, frog, ang, mariana, Cat, Dog, Bird, Lion, Frog, Ang, Mariana]
Feature Extractor C
  [cat, caT, cAt, cAT, Cat, CaT, CAt, CAT, ... , MARIANA]
Feature Extractor D
  [cat, dog, bird, lion, frog, Ang, Mariana]
Feature Extractor Ef
  [cat, dog, bird, lion, frog, cats, dogs, birds, lions, frogs, Ang, Mariana]
```

Depending on the sensitivity of the encrypted database and the desired degree of matching flexibility, one of the above feature extraction methods will be more appropriate then the others.
Without diving deeply into the discussion on feature extraction methods and trade-offs, it suffices to say that feature extractor E is a fairly good choice, as it allows us to query the database agnostic of capitalization or grammatical number while distinguishing between common and proper nouns.

2.0.2 Searching Over English Sentences

Consider the following set of simple sentences and their corresponding 1-grams.

```
‘War is peace.’
  [War, is, peace.]
‘Freedom is slavery.’
  [Freedom, is, slavery.]
‘Ignorance is strength.’
  [Ignorance, is, strength.]
```

Consider the results to the following two sets of 1-gram queries if simple keyword search is used directly.

```
Set A: [‘war’, ‘freedom’, ‘ignorance’]
Set B: [‘is’]
```

Only queries containing exact lexicographical matches against a previously inserted 1-gram will yield a positive search result. Consequently, the three queries in Set A will match nothing, while the query in Set B will match all three sentences.

In this case, preprocessing using standard NLP techniques like stemming and part of speech tagging can help normalize the capitalization and grammatical number of input terms as well as potentially removing common terms like auxiliary verbs, propositions, conjunctions etc.

Next, consider these slightly more complicated queries.

```
['peace is war', 'ignorance has strengths', 'war's peace']
```

Since our keyword query scheme operates on 1-grams, we can accommodate the above queries in one of several ways. The naive approach is to treat each query string as a single 1-gram. However, doing so will yield *no matches*. Alternatively, we can break each query string into corresponding sub-queries containing individual 1-grams, then apply a boolean operation on the subsequent results. However, this approach does not allow us to express any conditionality on the *ordering* of the queried terms. A better approach is to construct sub-queries containing "phrases" of the original query.

For example, from the search string *"peace is war"*, we can construct the following sub-queries: {*"peace is", "is war", "peace is war"*}. Doing so gives us the ability to carry out order-preserving partial matches using a single query string. We can then use the binary results from these sub-queries to calculate an overall match score. The desired matching behavior will directly affect how this score is calculated and interpreted.

This flexibility comes at a price. In order to match the sub-queries, all possible "sub-phrases" of the input corpora must be inserted into the encrypted database. However, as Section 3.3 shows, since the search time of a query over a single document is independent of the number of terms generated for that document, the SADS scheme can easily accommodate a large number of input terms with no performance penalty.

3 Secure Search

3.1 Secure Search Model and Requirements

We consider secure search as a solution for controlled data sharing in the following setting: a party that we call server possess a database and it would like to provide a group of parties that we call clients with access to its database through search queries. However, the database contains private information for the server and he wants to limit the information that will be revealed to the clients to only what is directly relevant to the queries that they are allowed to make in addition to controlling who is able to submit queries. At the same time the clients would like to protect the content of their query and even the fact that they are submitting a query that indicates their interest in the type of information contained in the database. In spite of the privacy concerns both clients and server have they still have incentive to execute such controlled search queries, which will enable them to find out whether they possess data of mutual interest that they can further exchange. A protocol that would achieve the desired functionality needs to provide the following **security guarantees**:

- *Query privacy* - the server does not learn information about the query.
- *Client's anonymity* - the server does not learn the identity of the querying client.
- *Database privacy* - the client learns only results matching its query.
- *Client's authorization* - the server can control which clients are allowed to search the database.

The *Database Privacy* requirement listed above assumes that there is a way to unequivocally determine the matching database entries for a given query. If we consider a query at syntactic level such as keyword search its the question for its matching context is well defined. However, if we want to interpret a query semantically, the meaning of "*matching*" becomes fairly complicated. As we will see in Section 2, overly restrictive interpretations of *matching* can result in semantically equivalent information not being matched by a query, resulting in a semantic false negative. Similarly, overly relaxed *matching* can cause unrelated documents to be returned in response to a broad or meaningless query (for example, the words, 'and', 'the', etc.), resulting in a semantic false positive. When engineering real-world secure private search systems that aims to identify relevant content, we need to consider both syntactic and semantic levels for interpretation of the query.

3.2 Secure Anonymous Database Search (SADS)

The starting point for our search system is a protocol for secure anonymous database search [10] that offers a model in which we achieve a balance between the conflicting requirements of security, anonymity and practical efficiency. In particular we consider computational complexity per query that scales linearly with the search entries in a document or the number of all possible search terms unacceptable. This rules out encrypted search protocols such as [3–5, 12], which provide query and database privacy but do not address the anonymity and authorization requirements.

The model of the SADS system makes necessary relaxations in strict cryptography security definitions to meet the minimal security requirements and utilizes distribution a limited amount of trust to two intermediary parties that mediate the sharing process between the client and database owner enhancing the privacy and anonymity guarantees for both participants. The two additional parties, *Index Server*(IS) and *Query Router* (QR), are viewed with neutral parties available to regulate the search process that are assumed to act semi-honestly in the execution of the protocol.

The roles of the Index Server and the Query Router in the search protocol are as follows. The server computes search structures from the its encrypted data which can be used to outsource the search functionality to the IS without allowing him to learn the actual document content. Although the IS will be able to compare the return results for different queries the anonymity guarantees for the client will prevent him from associating the queries of a particular client. The role of the QR is to protect the identity of the clients and at the same time enforce the authorization check on behalf of the server. The QR receives the queries from all clients and "blinds" their identities before submitting them to the IS and correspondingly returns the results to the client. However, the QR is not trusted to learn either the queries or the results. Thus the new security requirements with respect to the two new parties are as follows:

- IS - privacy for the Server's database, anonymity for Client.
- QR - privacy for the query and the return results.

3.3 Building Protocols

We use the following three main building block protocols to construct the SADS scheme (See [10] for details):

Re-routable Encryption

We introduce the notion of *re-routable encryption* which represents an encryption scheme that allows transformation of encryptions under different keys given corresponding transformation keys without leaking information about the encrypted messages. In the setting of re-routable encryption there is a party that is responsible for routing messages between senders and receivers utilizing the transformation between encryptions. Although the routing party is trusted to match senders and receivers it is not trusted to learn the routed messages. We also augment this construction with the option that the routing party forwards only partial information extracted from the transformed ciphertext. (See [10] for a formal definition of re-routable encryption)

PH-DSAEP+

While the re-routable encryption scheme provides a framework to realize the functions of the query router we need an additional property from the encryption scheme that will facilitate the efficient search at the IS. Since the standard cryptography definitions of security (e.g., [6]) require an encryption scheme to be probabilistic which makes sublinear search complexity impossible, we need to use deterministic encryption. This tradeoff of security for efficiency follows the idea introduced by [7], who define deterministic encryption in the public-key setting, and show how to convert a standard (probabilistic) PKE to a deterministic one. We follow the same approach, adapting it to the secret key setting. We construct the private key encryption scheme PH-SAEP+ as a combination of the Pohlig-Hellman function [9] and the SAEP+ (short for Simple-OAEP) padding construction introduced in [2].

Bloom Filters The private key deterministic encryption scheme PH-DSAEP+ provides search capability over ciphertexts that achieves sublinear complexity. In order to utilize this capability we need to construct

searchable tags from the deterministic encryptions. For this purpose we use Bloom filters [1] that extend the idea of hashing using multiple hash functions, which improves the collision probabilities. This facilitates efficient search that requires constant time per Bloom filter independently of the number of terms added, guarantees there will be no false negatives, and allows a tunable rate of false positives that can be bounded by a desired value by choosing appropriate parameters for the Bloom filter. For the purposes of our scheme we compute a Bloom filter (BF) per document in the database containing all keywords where we derive hash function indexes for the BF from the PH-DSAEP+ encryptions of the keywords.

3.4 Secure Anonymous Database Search

We utilize the building block protocols described in the previous sections to instantiate a protocol for secure anonymous database search for the architecture presented in Section 3.2.

SADS Construction

Given a server(S), a client (C), a query router (QR) and an index server (IS), we define secure anonymous database search (SADS) scheme (Figure 1) with the following algorithms:

- **Key Generation:** S chooses an encryption key. IS chooses an encryption key. A client C generates two keys for query submission and return result. To authorize C to search S, QR and C run a key exchange protocol to allow QR to obtain a ratio key between the S's encryption key and C's query submission key. Also IS, C and QR run a key exchange protocol so that QR obtains a ratio key between IS's encryption key and C's return result key.
- **Preprocessing:** S generates for each of its documents a Bloom filter from the encryptions of its words under PH-DSAEP+ under his key. S sends the resulting Bloom filters to IS.
- **Query Submission:** We instantiate the re-routable encryption protocol for query submission as follows: C encrypts his query with PH-DSAEP+ with his key and sends it to QR, QR re-encrypts the ciphertext with its transformation key for C, extracts Bloom filter indexes from the new encryptions and sends them to IS.
- **Search:** IS uses the obtained indexes to execute BF search to get the result R.
- **Query Return:** The query result is returned with a different instantiation of the re-routable encryption protocol: IS encrypts R with PH-SAEP+, sends it to QR, QR re-encrypts the ciphertext with the return result transformation key for C and sends it to the client.

For an argument of the security properties of the SADS scheme we refer the reader to [10].

4 General Flexible Private Secure Search Framework

Simple keyword matching alone is not always enough to satisfy real-world queries over complex, structured documents.To overcome the constraints imposed by SADS, we informally introduced the process of *preprocessing* both the document corpus as well as all incoming queries to achieve greater semantic awareness and improved search flexibility while operating under the exact match constraint. The examples in Section 2 focused on natural language documents. However, the general framework presented here goes beyond textual corpora, as the feature extraction process can be applied to any document format. We capture the idea for extracting such characterizing features of documents with the following definition of an *extractor* that can be instantiated with any specific feature extraction algorithm.

Definition 1 (Feature Extractor) *A feature extractor is an algorithm $FExt$ that takes as an input document D and returns a set of tokens $f_1, f_2, ..., f_n$, which identify the class in which D belongs according to the similarity metric implemented by the extractor.*

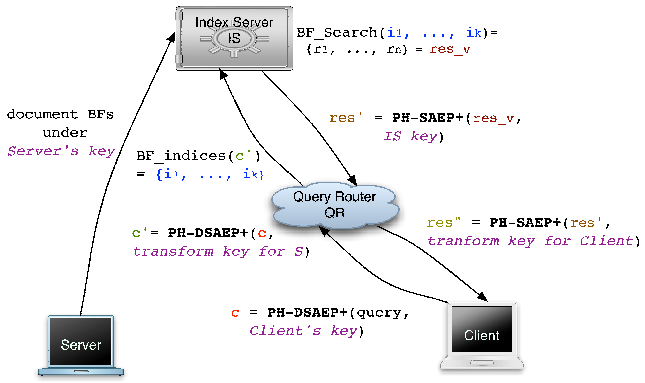


Figure 1: SADS query submission and result return

The feature tokens produced by various extractors can now be used for search and combined with any scheme that utilizes a search structure defined as follows:

Definition 2 (Search Structure) *A search structure B is any object for which there exist efficient algorithms to create an empty search structure B (***Init***(B)); to insert a token t in the search structure B (***Insert***(B,t)), and to query whether t is present in B (***Query***(B,t)).*

The exact efficiency for the $Insert$ and $Query$ will depend on the specific algorithms they implement. We proceed to define a document search scheme that instantiates the above structures with different document features:

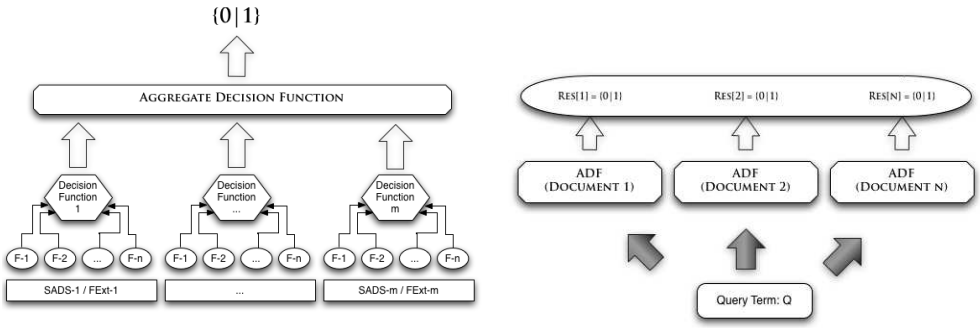
Definition 3 *A document search scheme S_{FExt} = (***Insert_Document***, ***Query_Token***) is defined by the following two algorithms:*

- **Insert_Document**(D_i) — *creates a search structure that contains as entries the features extracted from the document.*
- **Query_Token**(t) — *return all documents that have a feature matching the search token t.*

The above scheme allows search on documents with respect to a particular feature given by the underlying extractor. However, the similarity measure conveyed by just one feature of the query or one particular feature extractor may not be sufficient for a meaningful match. For this purpose we want to be able to combine the search results over several features of the query string and also over several different feature extractors. We create two frameworks that allow tunable definitions for the intended document matches for a query. First we translate the query string into multiple features and use the search results for all of them to compute a similarity matching coefficient for each document with respect to the query.

Definition 4 (Weighted Query) *Let S_{FExt} be a document search scheme. A* weighted search *computes matching coefficients for each document in the database based on their similarity to a string Q. We define an algorithm* **Weighted_Query**$_{S_{FExt}}(Q)$ *as follows:*

1. *Extract the searchable features of the query:* $\mathbf{F} = FExt(Q)$.
2. *Search for each feature* $f_i \in \mathbf{F}$ $\mathbf{R_i}$ = **Query_Token**(f_i).
3. *For each document D_j compute a vector $\mathbf{P}_j$ of the features for which it was a match.*



(a) Sample Query Over Single Document (b) Sample Query Over All Document

Figure 2: Sample Query

4. *Assign to each document D_j a matching weight given by* ***Decision_Function***$(\mathbf{P}_j)$ *, where the function* ***Decision_Function*** *can be instantiated with any function that assigns weights to a set of features extracted from a string.*

5. *Return a vector* **Res** *of the matching weights for all documents.*

The next level for extending the capabilities for similarity matches is to utilize several different feature extractors. We keep different semantic summaries of the documents given by search structures built with different feature extractors and combine the matching weights of the documents for each feature extractor to compute the final search result.

Definition 5 (Threshold Query over Multiple Featured) *Let $S_{FExt_1}, \ldots, S_{FExt_n}$ are document search schemes using different feature extractors, which have been initialized inserting the same set of document in each of them. A* threshold search *identifies all document that have aggregate similarity across all features to a string Q greater than a threshold values t. We define* **Threshold_Query**(Q, t) *as follows:*

1. *Compute the matching weights for the documents with respect to each feature* $\forall 1 \leq i \leq n$ $\mathbf{Res}_i =$ **Weighted_Query**$_{S_{FExt_i}}(Q)$.

2. *For each document D_j compute a vector $\mathbf{P}_j$ of the matching weights for each feature.*

3. *Return a document D_j in the result vector* **Res** *if* ***Aggregate_Decision_Function***$(\mathbf{P}_j) > t$*, where* ***Aggregate_Decision_Function*** *can be instantiated with any functions that computes a single weight value from the weights of the different features extracted from the documents.*

5 Proof of Concept Private Secure Document Database

Using SADS and the private secure search framework described in Section 4, we have engineered an efficient and usable secure private document database over two separate corpora; a collection of 106 RFC documents (RFC) and all verses of the King James Bible (KJB). To further demonstrate the adaptability of the proposed framework, we also present a similar document database capable of executing queries over Chinese novels. Three different feature extractors, **1-Gram**, **Stemmer** and **TF-IDF**, are used to create the final document database for both the RFC and KJB corpora. Section 5.1 describes the unique search capabilities each feature extractor provides. As we will show at the end of this section, adjusting the decision and aggregate

decision functions controlling the feature extractors defines the query matching behavior of the database. Therefore, choosing the right feature extraction methods along with suitable decision and aggregate decision functions to achieve the desired search behavior and efficiency can be seen as an optimization problem.

5.1 English Document Feature Extractors

5.1.1 1-Gram

The 1-Gram feature extractor is the simplest in the trio, and does simple word tokenization, returning a list of 1-grams. While extremely simple to compute, the 1-Gram feature extractor can only be used to execute order agonistic exact matches on single words. 1-Gram can be extended to N-gram which returns a list of terms up to N-grams. Each N-grams term is equal N consecutive words in the document.

5.1.2 Stemmer

Stemmer is a full blown NLP based feature extraction method. As the name suggests, Stemmer enables queries which are agnostic of language features like capitalization, verb tense and grammatical number. More specifically, Stemmer uses the NLTK library [20], and applies the following operations on the input document.

1. Paragraph, Sentence, and Word Tokenization
2. Part of Speech Tagging and Filtering
3. Stemming
4. Capitalization Processing
5. Sentence-Scope All N-gram Generation[1]

5.1.3 TF/IDF

The Term Frequency / Inverse Document Frequency (TF/IDF) feature extractor returns terms "unique" to each document with respect to the entire corpus. Each document in the corpus is partitioned into features containing up to N tokens (we use N = 3). The TF/IDF value is computed for each feature and sorted in decreasing order. The top M terms are output by the feature extractor. In our test results, we used $M = 1000$ for the RFC corpus and $M = 100$ for the KJB corpus.

5.2 Decision and Aggregate Decision Functions

5.2.1 Decision Functions

We present three decision functions, Decision-Function-[1,2,3], which are ranked by decreasing order of matching strictness. Each decision function implements $Decision\_Function(Q)$ as presented in Section 4, and returns a normalized score s such that $\{s : s \in R, 0 \leq S \leq 1\}$. Keep in mind that the decision function is used to query the set of features extracted by some feature extractor $FExt(D_i)$ on every document D_i in the document corpus.

Decision-Function-1 represents an exact match decision function which returns 1 if the longest term in the set of features extracted for a query is found, and 0 otherwise.

Decision-Function-2 is a relaxed version of Decision-Function-1 and returns 1 if **all** features extracted from the query is found in the particular document, and returns 0 otherwise. Depending on the feature extractor used, this decision function exhibits a range of different behavior. While the output score is still binary, Decision-Function-2 can be used to constrain the "in-order" requirement of the matching behavior. Suppose the query is a five word phrase, and the feature extractor used returns all 3-grams derived from query. Decision-Function-2 will return a match (score of 1) if all *three* 3-grams are found somewhere within the document. Clearly, the "in-order" requirement can be adjusted by changing the n-grams outputted by

[1] Yields all possible n-grams where n ranges from 1 to length of sentence

the feature extractor, where a value of n=1 represents the degenerate case where order is not considered. The experimental results shown in Figure 3(a) uses n=2 for this decision function on all feature extractors.

Decision-Function-3 is the most relaxed decision function, which computes percentage of features found in a particular document over all the features extracted from a input query string. Consider the vast range of matching behavior possible with Decision-Function-3, when used in conjunction with different feature extraction methods.

Aggregate Decision Function is instantiated with a linear threshold function that combined a weighted manner the scores produced by the searches with each of the three feature extractors. Both weights and threshold are parameters which can be changed to alter the matching behavior of our system.

5.3 Experimental Results

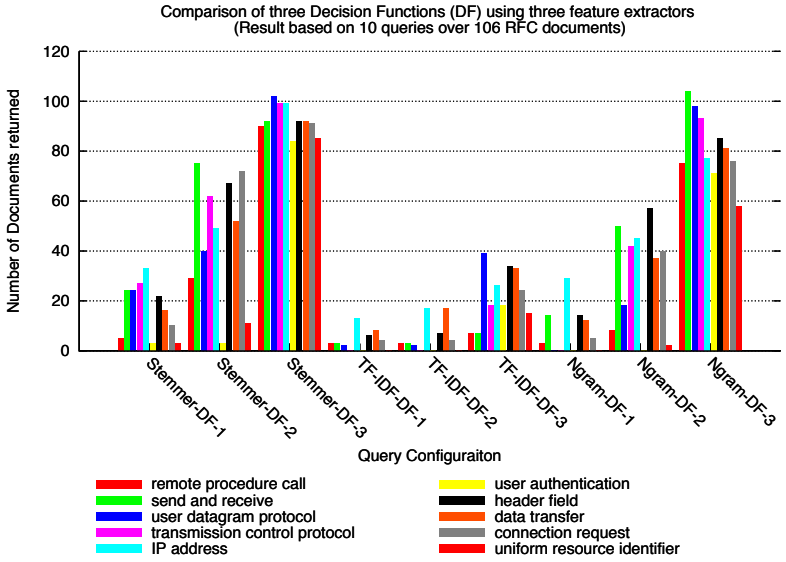
We queried 10 sample phrases over the demonstration database under various settings. Figure 3(a) shows the number of documents matching each query using each of the three decision functions described in Section 5.2.1 and each of the feature extractors described in Section 5.1 when the threshold value is fixed at 0.01. This demonstrates the matching behavior of each individual decision function when used in conjunction with each of the three decision functions. We observe that the number of results returned by the different decision functions increases in the same pattern when combined with each of the feature extractors. The results from the summary (TF/IDF) extractor are most conservative since it preserves a limited number of searchable terms per document. Most of the time the stemmer feature extractor returns more results than the N-gram since it increases the matches of a word to all other words with the same stem. Figure 3(b) demonstrates the decrease in the number of result documents when we increase the threshold value across all decision functions.

5.4 Trade-offs and Usability

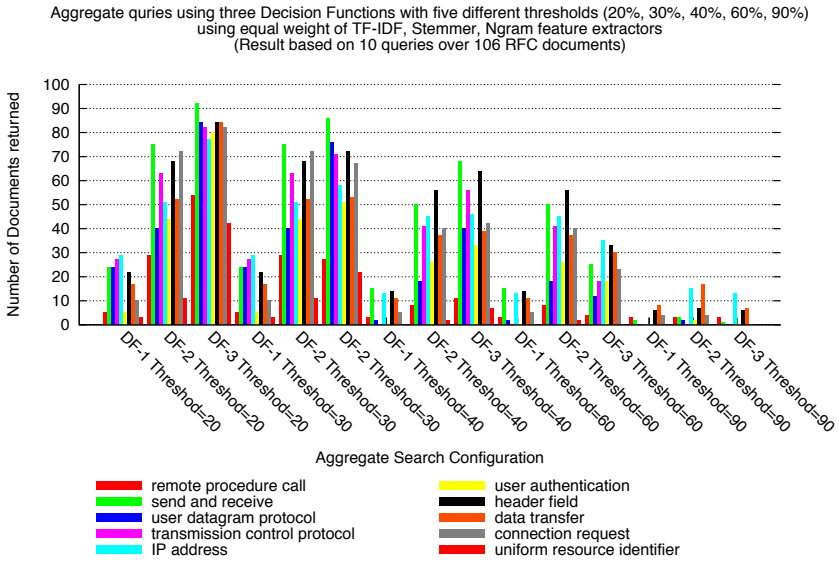
Looking at Figures 3(a) and 3(b) together, it is easy to see that different choices of feature extraction methods, decision and aggregate decision functions directly impact the matching behavior of the private secure search system. While the underlying matching mechanism remained the same across all test cases, the final query results exhibited large variations and alludes to some deeper trade-offs. As we have seen, when dealing with real-world documents rich in structure and semantic nuances, the measurement of false positive and false negative error rates become complex, context specific tasks. Our implementation incorporates three well-known and currently used decision functions, and produces the same results barring false-positive results added by the underlying Bloom Filter based implementation of the search. This is a tunable parameter, and can be reduced to negligible levels at the cost of storage space at the discretion of the system designer.

The system designer must consider the desired security and privacy requirements at a semantic level, and carefully consider the optimal tolerance for semantic false positive and false negative errors, which is often a non-trivial compromise. For example, should capitalization and grammatical number be ignored by the search engine? The answer is not obvious and depends upon user and designer intent. To analytically approach this question, we must consider several factors like the "cost" of a semantic false positive versus a false negative, the potential errors introduced by the algorithms used for feature extraction as well as their computational costs. Correctly tuning the matching behavior to comply with high level security and privacy requirements at a semantic level is perhaps the most important problem for any private secure search system.

Our proposed framework allows the system designer to break down this complex problem in independent subproblems under a simple to understand structure. Such an approach allows the designer to safely make incremental improvements to the system. More importantly, the proposed framework allows the same private secure search system to be easily parametrized to adapt to a wide range of usability and privacy requirements as well as different document corpora. At one extreme it can support full fidelity of existing search types with sufficient resources, and as it is based on the SADS system, search times remain dominated by network latency rather than any overhead introduced by the system. [10] It thus lends itself to the creation of practical real-world search systems.



(a)



(b)

Figure 3: Flexible Search Results

6 Conclusion

The SADS scheme enables exact keyword searching over an encrypted database providing maximal privacy guarantees under practical efficiency requirements. However, real-world use cases often demand more sophisticated query capabilities. We illustrate several common engineering pitfalls which can compromise the integrity of secure private information sharing systems. Since the class of logical flaws described in Section 2 are attributed to the agnosia of the format and content of the document corpus and not to the matching mechanics provided by SADS, we broadly distinguish the errors exhibited by our private information sharing system into to categories; syntactic errors and semantic errors. While syntactic errors, or errors caused by underlying keyword matching mechanism, can be easily quantified and prevented, semantic errors can be both difficult to detect and prevent.

To minimize semantic errors and to allow for greater query flexibility, we have created a framework for engineering secure private information sharing systems using cryptographic primitives like SADS. Using multiple unique semantically aware feature extractors in parallel and a two level document relevance decision scheme, our framework allows the system designer to control the degree of query flexibility as well as methods in which searchable information is extracted from the document corpus.

In order to demonstrate feasibility of our framework, we have engineered a private secure search system capable of executing complex queries over two distinct document corpora; RFC documents (*RFC*), and The King James Bible (*KJB*). Furthermore, as Section 5 shows, by adjusting several key parameters in the feature extraction process and the two level document relevance functions, we can control the flexibility of the query system. In other words, the decoupling of feature extraction and document relevance calculation from the underlying cryptographic mechanisms allows the system designer to implement a document matching policy engine independent of the secure encrypted search scheme used.

References

[1] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] Dan Boneh. Simplified OAEP for the RSA and Rabin functions. *Lecture Notes in Computer Science*, 2139:275–291, 2001.

[3] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Proceedings of EUROCRYPT'04*, pages 506–522, 2004.

[4] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows pir queries. In *Proceedings of CRYPTO'07*, 2007.

[5] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *the Theory of Cryptography Conference (TCC*, pages 535–554. Springer, 2007.

[6] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[7] A. Boldyareva M. Bellare and A. O'Neill. Deterministic and efficiently searchable encryption. In *Proceedings of CRYPTO'07*, 2007.

[8] Hyun-A Park, Bum Han Kim, Dong Hoon Lee, Yon Dohn Chung, and Justin Zhan. Secure similarity search. In *GRC '07: Proceedings of the 2007 IEEE International Conference on Granular Computing*, page 598, Washington, DC, USA, 2007. IEEE Computer Society.

[9] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms overgf(p)and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[10] Mariana Raykova, Binh Vo, Steven Bellovin, and Tal Malkin. Secure anonymous database search. In *CCSW 2009.*, 2009.

[11] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Eurocrypt'05*, 2005.

[12] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, page 44, Washington, DC, USA, 2000. IEEE Computer Society.

[13] `http://www.box.net/`. Box.net.

[14] `http://www.carbonite.com/`. Carbonite.

[15] `http://www.cx.com/`. Cloud experience.

[16] `http://www.dropbox.com/`. Dropbox.

[17] `http://www.evernote.com/`. Evernote.

[18] `http://www.mozy.com/`. Mozy.

[19] `http://www.navajosystems.com/`. Navajo systems.

[20] `http://www.nltk.org/`. Natural language toolkit.

[21] `http://www.sugarsync.com/`. Sugarsync.

[22] `http://www.vaultive.com/`. Vaultive.