Private Search in the Real World

Vasilis Pappas, Mariana Raykova, Binh Vo, Steven M. Bellovin, Tal Malkin
Department of Computer Science
Columbia University, New York, NY, USA
{vpappas, mariana, binh, smb, tal}@cs.columbia.edu

ABSTRACT

Encrypted search — performing queries on protected data — has been explored in the past; however, its inherent inefficiency has raised questions of practicality. Here, we focus on improving the performance and extending its functionality enough to make it practical. We do this by optimizing the system, and by stepping back from the goal of achieving maximal privacy guarantees in an encrypted search scenario and consider efficiency and functionality as priorities.

We design and analyze the privacy implications of two practical extensions applicable to any keyword-based private search system. We evaluate their efficiency by building them on top of a private search system, called SADS. Additionally, we improve SADS' performance, privacy guaranties and functionality. The extended SADS system offers improved efficiency parameters that meet practical usability requirements in a relaxed adversarial model. We present the experimental results and evaluate the performance of the system. We also demonstrate analytically that our scheme can meet the basic needs of a major hospital complex's admissions records. Overall, we achieve performance comparable to a simply configured MySQL database system.

1. INTRODUCTION

Encrypted search — querying of protected data — has come into the foreground with growing concerns about security and privacy. There are many variants of the problem that protect different things: the searchable data, queries, participant identities, etc. Existing schemes also differ in their expected operational environment. The majority of encrypted search mechanisms concern data outsourcing [4–6, 8, 13, 33, 34] and to a lesser degree data sharing [9, 17, 30]. Data outsourcing concerns the case where one party wants to store its encrypted data on an untrusted server and be able to search it later. Data sharing involves one party who provides limited search access to its database to another. These two settings require different privacy guarantees of an encrypted search system; data outsourcing is not concerned with protecting the data from the querier, since he is the owner. Furthermore, specific implementations may return different things (e.g., number of matches, document identifiers, related content, etc.) or may differ in number of participants, trust assumptions, anonymity requirements, revocation of search capability and other areas. All of these factors affect performance. Choosing a different definition of "sufficient" privacy can greatly affect inherent cost. Making the right choice, in accordance with the actual, rather than theoretical, threat model can lead to a very functional system, rather than one that is theoretically perfect but unusably costly in practice.

In this paper we step back from absolute privacy guarantees in favor of efficiency and real-world requirements. These requirements include not just what may leak, but to whom; depending on the particular practical setting there may be parties who are at least partially trusted. Our goal is to describe and build systems that meet the privacy guarantees matching the actual goals for a given scenario, so that we may improve efficiency. Towards this end, we present a set of generic extensions, applicable to any keyword-based private search system. We discuss the importance of each of these, the challenges for their secure implementation and analyze their privacy implications in terms of leakage. To evaluate their efficiency, we developed them on top of SADS [30], an efficient private search system that uses Bloom filters. In addition, we describe and implement a number of new features in SADS that improve its performance, privacy guarantees and functionality. Finally, we describe and analyze the performance of the extended SADS system in a real-world scenario, using health records.

Our implementation and the obtained empirical results are an important contribution of this paper from the point of view of evaluating the real usability of the proposed system for practical purposes. Although theoretical analysis asserts that a Bloom filter-based search should be efficient, it is unwise to rely solely on theory. If nothing else, complexity analysis says nothing about constant factors, and says nothing about unexpected bottlenecks. It matters little if an algorithm has n^3 exponentiations if n is reasonably small and the entire system is in fact I/O-bound rather than CPU-bound [24]. Similarly, Kernighan and Pike noted that "measurement is a crucial component of performance improvement since reasoning and intuition are fallible guides and must be supplemented with tools" [23]. Our work shows that — asymptotic behavior aside — our scheme is practical across a wide range of input sizes. Equally important, it shows the cost of different kinds of privacy. Neither conclusion is amenable to a purely theoretical

ACSAC '11 Dec. 5-9, 2011, Orlando, Florida USA
Copyright 2011 ACM 978-1-4503-0672-0/11/12 ...$10.00.

This material is based on research sponsored by Air Force under agreement number FA8750-09-1-0075. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force or the U.S. Government.

analysis.

The contributions of this work are:

- We present two practical extensions, namely Document Retrieval and Range Queries, that can be used on top of any keyword-based private search system.
- We improve an existing private search system (SADS) to provide better privacy, support data updates and become more robust.
- We implement all of the above and provide extensive evaluation results and a case study. Code, datasets and data inputs are available online at `http://nsl.cs.columbia.edu/projects/sads/` for similar systems to compare with.

2. BACKGROUND

2.1 Secure Anonymous Database Search

The secure anonymous database search (SADS) scheme [30] provides the following search capability: it allows a search client (C) with a keyword to identify the documents of a database owner/server (S) containing the keyword without learning anything more or revealing his query. For this purpose the architecture of the system involves two semi-trusted parties: index server (IS) and query router (QR), which facilitate the search. In summary the scheme works as follows: the database owner computes search structures for his database — a Bloom filter (BF) per document built from the encryptions of all words of the document. Each authorized client receives keys that he uses to submit queries and decrypt the results; the QR receives corresponding transformation keys for the queries of that client. To submit a query, C computes an encryption of his query and sends it to QR. QR verifies that the client is authorized, re-encrypts the query with the corresponding transformation key, computes and sends the BF indices obtained from the encryption to IS. IS performs search across the BFs it stores, encrypts the identifiers of the matching documents and sends them to the QR; QR transforms the encryptions and delivers them to the client, who decrypts them to obtain his search results (see Figure 1).

The original implementation of SADS also includes a couple of optimizations/features enabled by the use of BFs. First, storing the BFs in transposed order – called *slicing optimization* – minimizes the number of bits that need to be read during search. That is because only bit slices corresponding to specific indices are read during a query and not all the BFs. This approach has two main benefits. First, it has better cache behavior because it fetches each slice once and uses it for all the result vectors; second, in some cases it avoids reading several slice portions if the corresponding bits of all the result vectors have been zeroed out. In addition, SADS also supports *boolean queries*. One naive way to do this is to search for each term separately and union or intersect the results. However, BFs can more efficiently handle ANDs by combining indices into a superset, and ORs are handled in parallel by the slicing optimization.

2.2 Security Definitions and Relaxed Privacy Settings

The strongest security definition for a generic encrypted search scheme in the setting of data sharing guarantees that the querier receives only the matching results, while none of the other parties in the protocol learns anything. If we formalize this intuition by applying the standard simulation security notion of Canetti [7], what the definition captures is that a protocol is secure if the views of the participants in the real execution (namely their inputs, random inputs, outputs, and messages they receive) are indistinguishable from their views in an ideal execution where all parties send their inputs to a trusted party who computes the results and sends them back to the receivers.

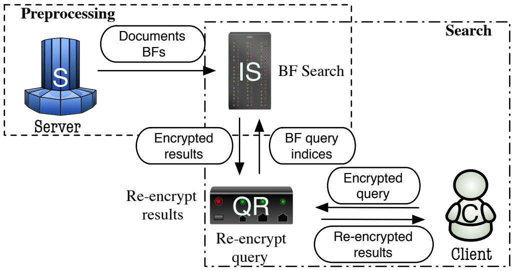

Figure 1: SADS overview.

Satisfying this level of privacy inevitably comes at efficiency cost. In many scenarios, weaker privacy guarantees may be sufficient. In particular, to achieve better performance, it is often acceptable to leak some controlled amount of information. The next definition gives the general security notion for encrypted search with certain privacy leakage.

DEFINITION 1. *A protocol π is a secure encrypted search protocol with privacy leakage $\mathcal{L}$, if for every real world adversary, there exists a simulator such that the view of the adversary in the real world execution, where he interacts with the honest parties, is indistinguishable from his view in an ideal world execution where he interacts with a simulator that takes as input $\mathcal{L}$ (and simulates the honest parties).*

SADS has the following privacy leakage $\mathcal{L}$ with respect to the parties that perform the search (i.e., the IS and the QR if they collude):

- *False Positive Database Leak*: a fraction of records that do not match the search criterion
- *Search Pattern*: the equality pattern of the submitted queries
- *Results' Pattern*: the equality pattern among the results
- *Similarity Database Leak*: search structures leaking information about similarity of data records.

In Section 5.1 we present a modification to the scheme that removes the last type of leakage that comes just from the search structures on their own.

The above security guarantees apply to the search functionality for exact match queries. The SADS search scheme further supports Boolean queries, which provide privacy guarantees for the non-matching part of the database, i.e., the querier does not learn anything about the non-matching records. With respect to the query privacy from the search party the Boolean queries reveal the matching pattern over the different terms in the search query in addition to the results' pattern. In Section 4 we introduce a scheme that realizes range query functionality that is based on OR queries and inherits the query leakage from the Boolean queries. The leakage from the OR queries in the context of the range queries means that given a range query the index server will be able to obtain identifiers for each logarithmic-sized sub-range that two records have terms co-occuring in, starting from the unique value and ranging up to the sub-range equal to half the size of the full range. It does not, however, learn what ranges these are, or, their size. The identifiers

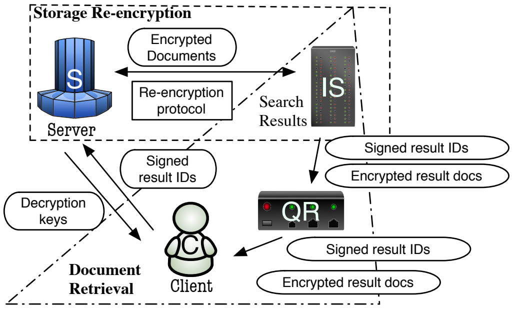

Figure 2: SADS with Document Retrieval.

can only be useful to determine patterns across multiple queries.

3. DOCUMENT RETRIEVAL

There exist many systems for searching databases to privately identify items of interest. An extension of obvious use is a system to then retrieve those items privately. One way to do this is with private information retrieval techniques, however these are very expensive, and can be even more expensive when fetching large numbers of records, or records of individually great size. We present a system that is much more efficient, at the cost of requiring a trusted third party, and can be modularly implemented to extend any private search system that returns handles representing matches.

Systems both with and without document retrieval have practical use. For example, a user may simply wish to establish that a server does have documents of interest to him, or may wish to determine how many are of interest, or learn about certain qualities concerning the data held there (subject to the search permissions granted by the server). Furthermore, even in systems that include document retrieval, separating this functionality from query is worthwhile. For example, the server may be running a paid service, and allow the user to operate in an initial stage wherein he determines what he wants, and a bargaining stage wherein they negotiate pricing, before purchasing the actual content.

Document retrieval poses its own challenge, especially when the data is not owned by the party retrieving it. In this scenario, returning additional data is a privacy leak for the data owner; at the same time, revealing the matching documents to the owner is a privacy leak for the retriever. Thus, the strongest security we would want to aim for would require us to touch the contents of the entire database [9]. This is a prohibitively expensive cost for applications that aim to work in “real time” over a large data set. One way to avoid this cost is to relax our security definition and allow leakage of the retrieval pattern (i.e. whether separate retrieval attempts touched the same documents). In the case of data outsourcing, this amount of privacy leakage easily suffices, since the untrusted server just searches for and returns the encrypted files that he stores to the owner who has the corresponding decryption keys [4, 8, 13]. This approach, however, is not applicable to the case of data sharing, where leaking the matching documents to the owner reveals more than the result pattern: he also knows the content of the documents, from which he can infer information about the query.

This problem is similar to that addressed by private information retrieval protocols (PIR) [10, 16, 29], wherein a server holds a set of items from which a user wishes to retrieve one without revealing which item he is requesting. It differs slightly in that we wish to retrieve multiple items (corresponding to the search results). It also differs in that we require that the selected set be certified and that the user does not learn content of documents outside of it. There are PIR schemes that address this [16], but at additional cost. Thus, our problem could be addressed by simply running an appropriate PIR scheme once for each document result. However, PIR is already quite expensive for a single document, and running them multiply would only aggravate this.

We address this by constructing a document retrieval scheme that can be used on top of any other scheme that returns document IDs. Our scheme maintains efficiency by introducing an intermediary party who stores the encrypted files of the database and provides the matching ones to the querying party. This party is given limited trust to perform the search, but he should not be able to decrypt the stored files. In this case we need to provide the querier with the decryption keys for the result documents; these are known to the data owner, who must be able to provide the correct keys obliviously without learning the search results. In Figure 3 we present a protocol that realizes the document retrieval functionality between a data owner (S) and a client (C) with the help of an intermediary party (P). For the purposed of this protocol we assume that there is a search functionality $EncSearch$ that returns the IDs of the documents matching a query from the client. For a query Q we denote $EncSearch(Q)$ the returned set of document IDs. The database of the server that is used for the protocol consists of documents $D_1, \ldots, D_n$. Our protocol also uses 1-out-of-n oblivious transfer (OT) functionality that allows two parties, one of which has input an array and the other has input an index in the array, to execute a protocol such that the latter party learns the array element at the position of his index and the former learns nothing. There are many existing instantiations of OT protocols, we use the protocol of [15], which allows best efficiency. The last tool for our constructions is an encryption scheme with the following property (defined in more detail in [30], which also gives an instantiation for such a scheme):

DEFINITION 2 (ENCRYPTION GROUP PROPERTY). *Let $\Pi = (GEN, ENC, DEC)$ be a private key encryption scheme. We say that Π has a group property if $ENC_{k1}(ENC_{k2}(m)) = ENC_{k1 \cdot k2}(m)$ holds for any keys k_1, k_2 and any message m.*

Intuitively, the security of this protocol is based on the secrecy of the permutation π, known only to P. Because it is not known to S, S cannot correlate the keys k'_{π_i} that are requested by C with the original indices of the matching documents. He learns only the search pattern of the querying party. We can take two approaches to mitigate this leakage. The querying party may aggregate requests for decryption keys to the server for the search results of several queries. Another solution is to extend the scheme to include additional keys pertaining to no real documents, which P can add to the sets of requested keys so that S cannot tell how many of the keys he returns correspond to query results. Step 2 of the re-encryption can be implemented using protocols for oblivious transfer [1, 11, 27].

4. RANGE QUERIES

We now present an extension that enables multi-dimensional range queries using any underlying private search system that, preferably, supports boolean queries in conjunctive normal form over exact string matches. We first describe our system as a general construction then discuss how some of the costs interact with the efficiency tradeoffs inherent in the SADS system due to the use of BFs.

4.1 General construction

This generic extension introduces the following additional costs

Storage Reencryption (*preprocessing phase*)

Inputs:
$S:\ D_1,\ldots,D_n$, keys $k_1,\ldots,k_n$ and $k'_1,\ldots,k'_n$;
$P:$ permutation π of length n ;
$S,P:\ (\overline{GEN},\overline{ENC},\overline{DEC})$ satisfying Definition 2
Outputs:
$S:\bot;\ \ P:\ \overline{ENC}_{k'_{\pi(i)}}(D_i)$ for $1\le i\le n$
Protocol:

1. S sends to P $c_i=\overline{ENC}_{k_i}(D_i)$ for $1\le i\le n$.
2. For each $1\le i\le n$ S and P execute 1-out-of-n OT protocol that allows P to obtain $k''_i=k_i^{-1}\cdot k'_{\pi(i)}$.
3. For each $1\le i\le n$ P computes $\overline{ENC}_{k''_i}(c_i)=\overline{ENC}_{k_i^{-1}\cdot k'_{\pi(i)}}(\overline{ENC}_{k_i}(D_i))=\overline{ENC}_{k'_{\pi(i)}}(D_i)$.

Document Retrieval

Inputs:
S : keys $k'_1,\ldots k'_n$;
P : permutation π of len n, $\overline{ENC}_{k'_{\pi(i)}}(D_i)$, $1\le i\le n$;
C : query Q;
S,P,C : search scheme *EncSearch* that returns IDs of matched documents to P,C.
Outputs:
S : cardinality of the output set $EncSearch(Q)$;
P : IDs of docs matching query Q from *EncSearch*;
C : the content of the docs matching Q from *EncSearch.*
Protocol:

1. S,P,C run *EncSearch* for query Q. Let $i_1,\ldots,i_L$ be the IDs of the matching documents.
2. P sends $Sign(\pi(i_1),\ldots,\pi(i_L))$ to C together with the encrypted documents $\overline{ENC}_{k'_{\pi(i_1)}}(D_{i_1})$, $\ldots,\overline{ENC}_{k'_{\pi(i_L)}}(D_{i_L})$.
3. C sends $Sign(\pi(i_1),\ldots,\pi(i_L))$ to S.
4. S verifies $Sign(\pi(i_1),\ldots,\pi(i_L))$ and returns $k'_{\pi(i_1)},\ldots,k'_{\pi(i_L)}$.
5. C decrypts $\overline{ENC}_{k'_{\pi(i_1)}}(D_{i_1}),\ldots,\overline{ENC}_{k'_{\pi(i_L)}}(D_{i_L})$ to obtain the result documents.

Figure 3: Protocol for Document Retrieval

over its underlying search system:

- For each inserted point, we will need to insert into the underlying search system $d\lg r$ terms, where d is the number of dimensions we are supporting and r is the size of the range of values supported per dimension.
- For each query, we will need to issue a boolean query using up to $2d\lg(\frac{q}{2})$ query terms to the underlying search system, where q is the size of the range being queried.
- The system presents repeatable unique identifiers for logarithmically cut sub-regions across all documents in a single query, and across multiple queries. If the underlying private search system does not guarantee full privacy of its queries, this can increase the information leakage over what would normally be incurred.

Our basic approach is to represent each ranged dimension as a binary value. Then, for each one, we create a strata for each digit of the value, and for each strata, divide the range into binary pieces according to the order of the digit, and assign each piece of each strata of each dimension a globally unique identifier. To insert a term to the search index, we insert the identifier of every piece that contains it (thus one term is inserted per dimension per strata, with a number of strata logarithmic in the size of the ranges). An example of inserting the value "11" using 4-bit numbers is hown in Figure 4 (top).

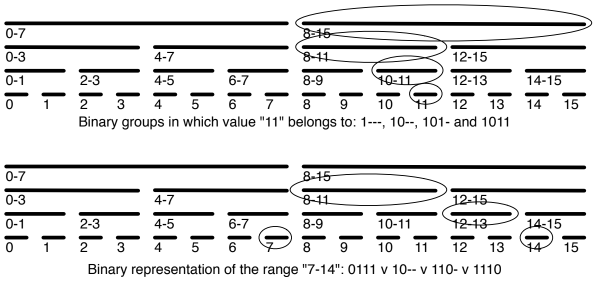

Figure 4: Terms used for inserting the value "11" (top). Boolean query for range "7-14" (bottom).

To issue a query, we create a boolean OR query. For each dimension, we start at the strata with the largest and least numerous pieces, and add to the query the largest piece that fits entirely within the query range. We iterate to lower strata, adding pieces that fit entirely into the range without covering values that are already covered by existing pieces in the query, and continue, if necessary, to the lowest strata which contains every individual value in the full range. We then create an AND query across all of the dimensions, resulting in a query in conjunctive normal form. An example query on one dimension is shown in Figure 4 (bottom).

Since every single piece of every strata that contains the representative value has been added to the index, this query will return true if and only if the range query contains it. The worst case query, is for the query range to straddle the midway point of the full range. This results in taking $2\lg(\frac{q}{2})$ query terms per dimension.

THEOREM 1. *A contiguous range query on a single dimension cannot require more than $2\lg(\frac{q}{2})$ disjunctive terms.*

PROOF. We begin with an initial lemma: a contiguous query cannot require more than two terms in a single strata, one in its lower half and one in its upper half. Let us assume to the contrary that it did require two terms within a single bisection of its range. Then, starting from the uppermost term, the range contains a subrange equal to at least four times the size of the elements of the strata (two in each bisection). Since the strata above uses elements of twice the size, and there is at least one term within the range that is not one of the endpoints, that term is a subset of a range from the upper strata that is contained entirely within this subrange. Thus, the term representing that range could have been chosen instead to replace two or more terms representing smaller ranges, a contradiction.

Given that each strata uses ranges twice the size of the strata beneath it, it is trivial to show via summing that a query cannot require terms from more than $\lg(q)-1$ strata. In conjunction with our lemma, we thus show that a contiguous range query cannot require more than $2(\lg(q)-1)=2\lg(\frac{q}{2})$ terms. □

4.2 Bloom filter construction

If the underlying system is based on Bloom filters, like SADS, we can describe the tradeoffs listed in the general construction in terms of increased Bloom filter size. The standard Bloom filter equation demands that in order to store n items with a false positive rate of p, our filter needs to have size m, show in formula (1). There are two factors that increase the necessary size of Bloom filters we must choose in order to maintain the same false positive rate. First, for every value inserted, there are now $d \lg r$ terms added, giving an increase in "effective" number of values for purposes of calculating proper sizes. Second, for every query, there are $2d \lg(\frac{q}{2})$ queries in CNF, any of which could be a false positive. If we assume in the worst case that a single false positive from a sub-query will cause a complete false positive, then we can give an upper bound on the multiplicative increase of false positive rate as $2d \lg(\frac{q}{2})$. Thus, the total size of the Bloom filter to ensure that the false positive rate does not exceed p is given by formula (2).

$$m = \frac{-n \ln p}{(\ln 2)^2} \text{ (1)} \qquad m = d \lg r \frac{-n \ln \frac{p}{2d \lg(\frac{q}{2})}}{(\ln 2)^2} \text{ (2)}$$

For practical purposes, this is a very reasonable increase in size, considering that most range query applications deal with orders of magnitude fewer values per record than exact string match queries, which may be used to index every single word in a tens-of-thousands-of-words long document.

An issue of greater concern is the magnification of existing privacy concerns, especially if we are using a system like SADS, which does not guarantee full protection of the result patterns. Because our construction will query the same sub-regions across multiple records in a query, and across multiple queries, if the result privacy is not protected against the server, he may be able to learn about the values stored within over time. For example, if a server sees that during a range query, two records had the same positive result for the same sub-region, it knows that they at the very least share a value in the same half-region (the largest possible sub-region). If over the course of multiple queries it sees those two documents match for a second sub-region, it then knows that they at the very least share a value in the same quarter-region. Over time, and seeing a sufficiently varied number of queries, it may learn exactly which documents share specific values. To prevent this, the ranged values could be re-indexed regularly based on the frequency of range queries being issued.

This is partially mitigated in the multi-dimensional case, since sub-regions of different dimensions cannot be differentiated, lending some additional obscurity. These are further obscured in systems like SADS where the ranged queries are interspersed with other types of queries including straight string matches and boolean queries. There is nothing to indicate to the holder of these identifiers what ranges they correspond to, or even if they are ranges at all. The quantitative evaluation of this reduction of information would depend on the nature of the records and their searchable attributes as well as the distribution of the queries that will be submitted. Therefore, the assessment of the significance of this leakage has to be done with respect to the specific data that will be used in the search scheme as well as the expected types of queries.

5. SADS SPECIFIC

5.1 Multiple Hash Functions

The BFs of different documents in the SADS scheme [30] share the same hash functions, and thus, the same BF indices for identical keywords. This is exploited by using a bit-slicing storage structure to improve query time. However, this has clear consequences for privacy:

- Due to commonality of indices for shared keywords, the search structures leak information to the IS about the similarity of the corresponding documents.
- The false positive rate of a single Bloom filter — the probability that a search query is matched incorrectly by it — with n bits, k hash functions, and m entries is $FP_{single} = (1-(1-\frac{1}{n})^{mk})^k$. If the false positive probabilities across different Bloom filters are independent, then the expected number of false positive results in a database with N documents is $FP_{single} \cdot N$. However, in the given situation, the false positive rates are not independent if documents share keywords. Let D_1 and D_2 be two documents where p fraction of the words in D_2 are also in D_1 and the query w is a false positive for the Bloom filter for D_1. The probability of a bit in BF_{D_2} to be set to 1 is $p + (1-p)(1-(1-\frac{1}{n})^{mk}))$ and therefore the probability D_2 has a false positive (all k search bits of w are set to 1) is $(p + (1-p)(1-(1-\frac{1}{n})^{mk})))^k$, which tends to 1 as p tends to 1.

We can avoid these issues by using different hash functions for the Bloom filters of each document. The BF indices for an entry would not be derived from its PH-DSAEP+ encryption but instead from keyed hashes of said encryption.

We implemented the *multiple hash functions* feature by generating a group of hash functions using a family of 2-universal hash functions [26]. In our implementation, we used HMAC over MD5 and SHA1 (using the document's ID as key) to generate BF hash functions, where the i-th hash function was $H_i(w) = H_1(w) + (i-1)H_2(w) \bmod P$, where P is a prime, $H_1(w)$ is HMAC(SHA1, ID, w), $H_2(w)$ is HMAC(MD5, ID, w) and w is the encrypted keyword.

5.2 Database Updates

So far we have assumed that the server's database does not change. It is preprocessed once in the beginning and from that point on the same data is used to answer all queries from the clients. However, in many practical situations the data of the server changes dynamically, which should be reflected correspondingly in the query results returned. The naive solution to run the preprocessing preparation of the database each time it changes, brings prohibitive efficiency cost. We would like to avoid processing each record in the database for updates that affect a small fraction of it. From a different point of view, though, the updates of the database can be considered private information of the server and thus the information about what records have been changed is a privacy leakage to any other party (in our case to the IS who holds the Bloom filter search structures). This type of leakage comes inherent with the efficiency requirement we posed above — if the update processing does not touch a record, clearly it has not been modified. Therefore, we accept the update pattern leakage as a necessary privacy trade-off for usable cost of the updates.

Now we look at the specific information that changes at the IS in the SADS scheme, and consider whether it has leakage beyond the update pattern of the documents:

- **Bloom filters**: As we discussed before, if we use the same hash function for the Bloom filters of all documents, then the search structures reveal the similarities between documents. In the case of an update this would be indicative to what fraction of the content of the document has been changed. If, however, each BF has a different set if hash functions, the update of a document would also include a selection of a new set of hash functions for its BF as well. The only information that the IS could derive from the update will be the change of the length of the document based on the number of 1's in the BF. However, this information can be obtained also from the length of the encrypted document that the IS is storing. In both cases, we can eliminate this leakage by padding the documents.

- **Encrypted documents** — Each encrypted document stored at the IS index is re-encrypted with a key that the IS obtains in an oblivious transfer execution with the data owner. If an existing document is modified, and the server encrypts it with the same key, then the IS can also use the same re-encryption key. If the server is adding a new document to his database, though, he should generate a new key of type k" (i.e., the encryption keys that the documents are encrypted with after the re-encryption of the IS). The guarantee that we want to provide is that the server does not know the permutation image of the key k'' that is used in the re-encryption of the document. We also want to avoid executing oblivious transfer for each document, which results in a complexity greater than the database size. Each permutation over $n + 1$ elements can be presented as the product of a permutation over the first n of the elements and a transposition of one of the n elements and the last unused element. Thus, it is sufficient to execute a protocol where the IS obtains re-encryption keys for the new document and for a random document from the rest. Intuitively this guarantees that the new re-encryption key could be assigned to any of the old documents or the new one, and if it was used for a previously existing document, then the new one receives the re-encryption key that was released from that document.

5.3 Optimizations

During the preprocessing stage, for each database document a Bloom filter containing its keywords is generated. In the SADS scheme, adding a keyword to the BF of a document involves encrypting the keyword under the server's key. Thus, preprocessing documents containing the same keyword incurs repeated effort. In order to avoid this unnecessary preprocessing cost, we can cache the BF indices for keywords. This avoids some recomputation, but requires additional storage space. Whether to do this, and how much to cache, depends on the nature of the documents and repeat frequency. This is also applicable in the case when multiple hash functions are used where the preprocessing of a keyword is not identical but shares a common and expensive intermediary result that can be reused. The caching capability we implement uses LRU removal policy.

In addition, SADS preprocesses each item of the dataset independently (i.e., computes the BF search structure for it), and furthermore, it handles the elements of each item separately (each word/value is inserted into the Bloom filer after a cryptographic transformation). This computational independence makes for simple and robust parallelization. The search phase, especially when using multiple hash functions, also permits parallelization of the computation of the search indices for a query. We used the open source Threading Building Blocks library [21] to implement the parallelization optimization. It is easy to use and well-integrated with C++. After analyzing the source code we found out that there is just one integer counter that we need to synchronize among the different threads: the Bloom filters counter. It took roughly 10 lines of code to parallelize the entire preprocessing phase – similar for the search phase too.

6. EVALUATION

To evaluate the practicality of our proposed extensions we implemented them in SADS (roughly 4 Klocs of C++ code in total) and we performed a number of measurements using realistic datasets: (i) the email dataset that was made public after the Enron scandal [31] and (ii) a synthetic dataset with personal information for 100K persons. The Enron dataset consists of about half a million emails with an average size of 900 bytes after stemming . During the preprocessing phase of SADS, a distinct Bloom filter for each email was created. Then, each of the email files was tokenized and the tokens where stored in the corresponding Bloom filter, after they were properly encrypted. The format of the second dataset is more close to a database than a collection of documents. Its schema consists of a single table with 51 attributes of three types: strings (first name, last name, etc.), numbers (height, SSN, etc.) and file links (fingerprint, private key, security image, etc.) and it is stored in a flat CSV (Comma Separated Value) file. The total size of that dataset, along with the files pointed in the records, is 51GB and the average size for a record is 512KB. During the preprocessing phase we created a distinct Bloom filter for each record and each of the attribute values where inserted after it was prefixed with the attribute name ("name_value") and properly encrypted. In both cases, we configured the BF parameters so as the false positive rate would be less than 10^{-6}.

The experimental evaluation setup was comprised by two servers and a client laptop. The servers had two four-quad Intel Xeon 2.5GHz CPUs, 16 GB of RAM, two 500 GB hard disk drives, and a 1 Gbit ethernet interface. The laptop was equipped with an Intel Core2 Duo 2.20GHz CPU, 4 GB of RAM, a 220 GB hard disk drive, and a 100 Mbit ethernet interface. All of them were connected through a Gigabit switch; they all ran a 64-bit flavor of the Ubuntu operating system. QR and IS were running on each of the servers, the queries were performed from the laptop. When Document Retrieval was enabled, the File Server was running on the same host with the IS.

6.1 Memory Consumption

Along with the timing measurements, we also monitored the memory consumption of the extended SADS system to determine scaling limits. We found out that the only significant factor was the type of Bloom filter storage. Bloom filters are stored either sequentially in a flat file or transposed using the slicing optimization. In the sequential storage case memory usage was constant; it grew consistently with the dataset size in the slicing case, because the structures are kept in memory and written to files at the end. During the search phase, both the client and the QR used a small, constant amount of memory (∼2MB). On the other hand, the IS's memory usage grew with the dataset size. In the sequential storage case, the file was `mmap`'ed; the amount of memory used was the Bloom filter size in bytes times the number of BFs (e.g. 1KB * 50K = 50MB). When the slicing optimization was enabled, we saw higher memory usage, ∼109MB for the same dataset. That was most likely due to the extensive use of C++ vectors, which we can further optimize in the case of much larger databases where the available RAM may become an issue.

6.2 Implementation Optimizations

We performed experiments using variable-sized subsets of both datasets while changing the size of the cache. As for the Enron dataset, we show that a good cache size is 5K keywords. This gives us a ∼90% hit ratio, while reducing the preprocessing time for 50K emails from 2h to 10m. Performing the same experiments for the synthetic dataset yielded slightly worse results, as some attribute values are unique. However, using a 10K keywords cache the hit ratio was 50% on the full dataset, which still is a significant gain.

We measured the speedup of the preprocessing phase on the full datasets, while increasing the number of threads. As we expected, the speedup grew linearly until the number of threads reached the number of cores in our servers – that is eight. When the number of threads was more than the CPU cores, the speedup slightly de-

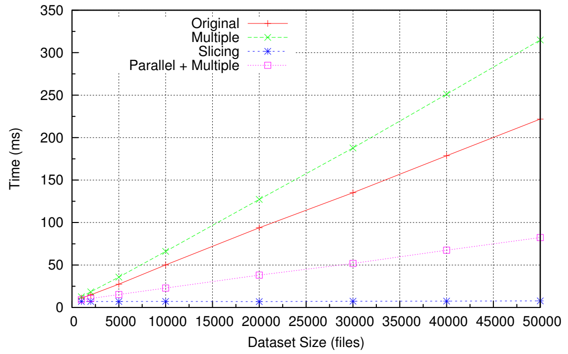

Figure 5: Average query time under different SADS configurations using the Enron dataset.

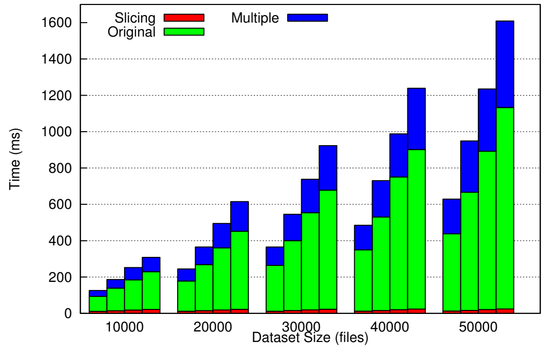

Figure 6: Average OR-query time under different SADS configurations using the Enron dataset. Each cluster is for a different dataset size and each bar is for a different term count (from 2 to 5).

clined, most probably due to thread scheduling overhead. Performance results for the parallelized search phase are presented in the next section.

6.3 Search Performance

The introduction of the multiple hash functions feature in SADS poses a trade-off between efficiency and privacy. Not only because of the higher computation overhead it adds but also because it is incompatible with the slicing optimization. In this section we explore in detail the effects of the multiple hash function scheme and also how parallel search could help amortize some of the performance penalty.

Figure 5 shows the comparison for four different configurations of SADS: (i) original, (ii) original with the slicing optimization enabled, (iii) using multiple hash functions and (iv) using multiple hash functions and parallel searching together. The search time reported in this figure is the total time elapsed from the point when the client issues the query to the QR until it receives the set of matching document IDs if any — no document retrieval. As expected, the average query time grows linearly using the original SADS configuration, as the actual search is done linearly over all the Bloom filters. Next, we can see that the slicing optimization greatly reduces search time to a point that it seems almost constant across different dataset sizes. Using the multiple hash functions feature we do get better privacy guarantees, but at the cost of increased search time by another factor that is proportional to the dataset size. That is because for each document we have to recalculate the hash functions and recompute the Bloom filter indices. Finally, we see that taking advantage of the commonly used multicore architectures does increase the performance of the search in the multiple hashing scheme. More precisely, the speedup when we used 8 threads on our 8-core servers was from 1.3 to almost 4 for the dataset sizes shown in the Figure 5. Thus, although the multiple hash functions feature increases the computation factor, we can amortize a great part of it by executing it in parallel. It is also worth noting that the multiple hash functions plus parallel searching configuration provides better performance that the original configuration, while on the same time it improves the privacy guarantees.

Column	Statistics min/ avg/ max	Query Time msec (stdev)	# of Ranges # (stdev)
Age	2/ 38/ 95	1,529 (244)	4.3 (0.6)
Height	58/ 67/ 78	959 (123)	2.8 (0.4)
Weight	90/ 175/ 280	1,979 (345)	5.8 (0.9)
SSN	1M/3.9G/6.5G	12,783 (805)	38.0 (2.4)

* the time for a single keyword query is ~400 msec

Table 1: Range queries timing on integer attributes of the synthetic dataset (100K records – 51GB).

Next, we evaluate the performance overhead of the multiple hash functions in boolean queries, and more precisely OR queries. To optimize the normal case – i.e., when the slicing optimization is not enabled – we skip BFs that already contain one of the search terms. That way we avoid searching over and over on Bloom Filters that already match the OR query thus reducing the overall searching time, especially when the search terms are frequent. Figure 6 shows the search time for OR queries under different SADS configurations. Each cluster of bars is for a different dataset size; each bar is for a different term count in the boolean OR query. The first bar is for two terms, the second for three, and the last two for four and five, respectively. The fact that the search time in each cluster grows sub-linearly to the number of terms clearly shows the performance gain.

6.4 Range Queries

The implementation of the range queries extension on top of SADS translates a range to a variable-sized OR query. In the average case, the number of the terms in the OR query depends on the size of the numbers in the range and the size of the range itself. To evaluate the practicality of that approach, we measured the time for performing range queries over the numeric attributes of the synthetic dataset. These are *age*, *height*, *weight* and *SSN*. The range of the values of these attributes relative small, except for the SSN which spans from one million to a few billions (first column of Table 1). For each of the attributes, we calculated ten ranges that each match about $1/10$ of the dataset. SADS was configured to use multiple hash functions and the parallel search optimization was enabled. Table 1 shows the average query time over the ten queries for each attribute, along with the average number of binary ranges that each query was translated to. In most of the cases, where the ranges are translated to a few binary ranges, the average range query time is low enough to be considered practical. On the other hand, the SSN attribute demonstrates the disadvantage of our range queries extension when dealing with big values. Still, the performance is not prohibitive, but, clearly, our range query extension yields better results for values that range from a few tens to a few thousands.

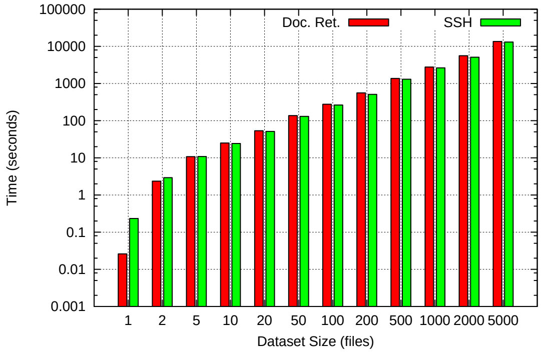

Figure 7: Average time for retrieving documents anonymously, compared to retrieving them non-anonymously using ssh file transfer. Average size of files being transferred was 27.8 Mb

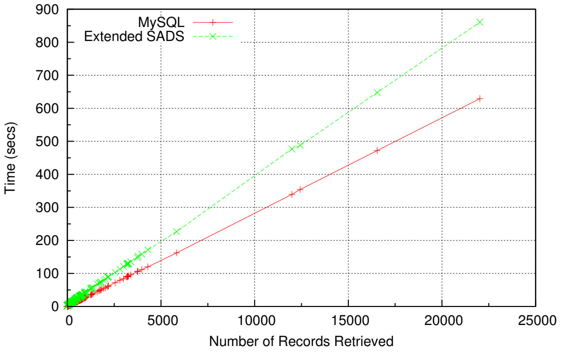

Figure 8: Comparison between the extended SADS and MySQL.

6.5 Document Retrieval

We implemented document retrieval using PH-SAEP and standard RSA signatures to sign query results. Using PH-SAEP puts a (likely over-restrictive) limit on the length of plaintext values. To handle this, we encrypt larger files using AES private key encryption, and store the key encrypted with PH-SAEP as a header in the encrypted file. The files can thus be read by decrypting the header with the appropriate PH-SAEP key and using the result to decrypt the content of the file. We preprocess the files in a way that provides an intermediate party with AES encrypted files under different AES keys and encryptions of these AES keys under some permutation of the keys k_1”, $\ldots k_n$”. The client will receive as results from the intermediary party the encrypted files, the encrypted AES keys, and the indices of the keys k” used for their encryptions. When he receives the decryption keys k” from the server, the client will first decrypt the AES keys and then use them to decrypt the remainder of the files.

Figure 7 shows the average time to retrieve documents using our scheme versus the number of documents being retrieved. This is shown in comparison to a non-privacy-preserving SSH-based file transfer. As we can see, our scheme adds very little overhead compared to the minimum baseline for encrypted transfer. The time also shows linear growth, suggesting that it is dominated by file encryption and transfer, rather than for the encryption and verification of the results vector itself.

As a point of comparison, Olumofin and Goldberg [29] present some of the best implementation performance results currently published for multi-selection single-server PIR. In their performance results, we see response times per retrieval ranging from 100 to 1000 seconds for retrievals of 5-10 documents on database sizes ranging from 1 to 28 GB. Our scheme scales strictly with number and size of documents retrieved, and not with the total database size. They do not state the sizes of the blocks retrieved in their scenario, but if we were to give a very high estimate of 1 MB per block, and assume they fetched 10 blocks every time, one could expect in our system that each query would take .7 seconds, still orders of magnitude short of the 100s fastest time they report for a 1GB database, and it would not scale up with increasing database size as theirs does, thus significantly beating the 1000s time they report for a 28GB database. Note that their system is not designed to protect privacy of the database, only of the request. The work of [14] presents a protocol for privacy-preserving policy-based information transfer, which achieves privacy guarantees weaker than SPIR and similar to ours. Direct comparison between our and their performance results is hard — they present timings only for the computation time without communication, which grows linearly with the size of their database. The maximum size of their database is 900 records with 2000 ms computation per record retrieval, while for our scheme the entire record retrieval time (computation plus communication) for a database with 25000 records is about 40ms (Figure 8, described in the next section).

6.6 Overall Performance

Finally, we compare the performance of the extended SADS system with a real world DBMS, MySQL. In order to do that, we implemented a SQL front-end for SADS that could parse simple conjunctive and disjunctive queries. Then, we loaded the synthetic dataset to both systems and we executed a number of queries of variable result set size. SADS was configured to use multiple hash functions and document retrieval was enabled. Parallel searching was also disabled, which means that we compared using the less efficient version of the extended SADS. Figure 8 shows the total duration of both the query and the retrieval of the data for our system and MySQL. Our scheme performs just 30% slower on average than MySQL, which is the price for the privacy guarantees it provides.

6.7 Case Study: Sharing of Health Records

We next examine, from a very high level, the suitability of our scheme for a hospital's admissions records database. (A database for full medical record storage is vastly more complex and is not addressed here.) A patient's health record is very sensitive information, usually kept by the patient's medical institution. There are cases, though, where such information needs to be shared among different institutions. For example, a patient may need to visit a different institution due to an emergency, or to get a second opinion. This sharing may need to be done privately for several reasons. In an emergency, a doctor may need to query all the institutions that share records with his without revealing the patient's identity, especially to the institutes that do not have information about him. If the querying is not private in that case, some institutions would learn information about a patient that has never visited them. Or, a patient may not want his institution to know which institution he visits for specialized needs, such as drug rehabilitation, so again the query for his record has to be performed privately.

A database of health records is similar to the synthetic dataset we used in our evaluation. It contains some searchable fields like name, date of birth, height, etc.; each record may be linked with several medical exam results like x-rays, electrocardiographs, magnetic tomographies, etc. In 1988, there were about ten routine tests during the hospital's admission process alone [20]; today, about thirty individual tests are done.[1] Taking into account that some of the results can be a few tens of Mbs — for example, a head CAT scan is about 32 MB — each health record could be a couple of hundred megabytes. One major hospital complex admits about 117K inpatients per year[2]; to a first approximation, their database would thus have several hundred thousands rows and 30–40 columns.

We have already seen, though, that the extended SADS scheme we propose can successfully handle a database of this size. Our evaluation demonstrated that document retrieval adds only a small overhead compared to simple transfer, thus easily scaling with the size of the document retrieved. Also, searching over 100K records with 51 searchable attributes each takes less than half a second, thus meeting real-world requirements. Finally, the support for updates in health records is a requirement covered by our extended SADS scheme. We conclude that our scheme is able to handle the requirements of this hospital, while preserving patient privacy.

7. RELATED WORK

Most of the existing constructions providing encrypted search capabilities aim to solve the case of database outsourcing [4–6, 8, 13, 33]. In this setting a party outsources the storage of his database to an untrusted server and wants to enable the server to execute searches on his behalf without learning information about either the data or the query. Unlike the data sharing scenario that we consider, this setting does not impose privacy requirements for the data with respect to the querier. A common technique in encrypted search schemes [4, 33] is to use trapdoors derived from query terms that enable the server to determine if a ciphertext matches the specific term. This implies the search complexity will be at best linear in the number of searchable tokens. A different approach in the setting of database outsourcing is to use inverted indices, where the search structures directly map all possible search terms to matches [8, 13]. Search then consists of finding the appropriate entry in the search structure for a given query's trapdoor. Such solutions leak the search pattern across a sequence of queries and are not easily extendable to allow more complicated queries beyond exact match when we need to preserve the privacy of the database from the querier.

Protecting the search pattern imposes efficiency costs. Bellare et al. [25] showed that in order to achieve sublinearity of the search complexity over encrypted ciphertexts, deterministic encryption is required, which leaks the search pattern. The works of [30] and [17] combine the idea of using deterministic encryption with Bloom filters [3] as search structures. However, the Bloom filter search structures constructed in these works leak the similarity of the underlying documents to the party who uses them for search. The work of [12] offers a scheme that exchanges search pattern leakage for efficiency improvement. While the suggested approach achieves sublinearity of the search complexity in terms of the number of searchable records, using preprocessing that transforms searchable tokens occurring in multiple records with unique tokens per record, it still requires time linear in the number of all searchable tokens contained in the matching records. Thus this solution is appropriate for scenarios with small numbers of searchable tokens per record, its efficiency improvements do not suffice in the case of long documents that contain many searchable keywords.

Search capability beyond simple exact matches has been achieved through constructions for attribute-based encryption [2, 19] and predicate encryption [22]. These approaches have a similar flavor to some of the searchable encryption schemes in the sense that they allow decryption only if the encrypted message satisfied a certain condition, which can be expressed, for example, as a dot product, Boolean formula or polynomial evaluation. But this also brings the related efficiency overhead that requires linearity in the size of all searchable tokens. Range queries are another type of queries with important practical applications. The work of [6] presents a protocol for range queries that comes with overhead $O(\sqrt{n})$ where n is the size of the domain of the searchable values. Shi et al. [32] incur $O((\log n)^D)$ computation overhead for D dimensional queries. Both of these schemes require that the token for the searchable interval is issued by the owner of the secret encryption key, which suffices for data outsourcing solutions but does not address the case of the data sharing.

If we consider the document retrieval functionality when the querier is the data owner, the search party can return the encrypted matching documents for which the querier has decryption keys. However, this approach is not applicable when the data owner and the querier are different parties and we want to hide from the data owner which documents were returned as results to the query. If the querier already knows the IDs of the documents of interest for his query the functionality of a symmetric private information retrieval (SPIR) scheme [16] when instantiated with the whole content of the database would theoretically allow the querier to retrieve the desired documents without the owner finding out what documents were retrieved. However, the efficiency cost is quite high. The PIR schemes that guarantee privacy for the query but do not provide privacy for the database already incur substantial efficiency overhead. Implementations of PIR were presented in [18, 29], and the work of [28] uses PIR techniques to provide partial privacy for SQL queries.

8. CONCLUSIONS

When we consider the question of secure search in practical settings, the privacy guarantees of a scheme are no longer the only relevant issue: a perfectly secure scheme that no one can use provides no actual privacy. The efficiency of an approach becomes a major factor in determining its usability given the available resources.

We adopted the relaxed security model of the SADS scheme; we extended its functionality by constructing a document retrieval protocol that runs in time proportional to the size of the returned set of documents and by providing range queries over integer data at a cost comparable to simple keyword queries in the average case. Both extensions take no advantage of any specific feature of SADS, making them applicable to any keyword-based private search system. Additionally, we improved SADS by: (i) providing a protocol that facilitates database updates without requiring processing of the whole database, (ii) using different hash functions for different BFs which provides better privacy guarantees and (iii) developing two implementation level optimizations, parallelization and caching.

The experimental results for the extended SADS system demonstrate its practicality: we achieve search and document retrieval time which is on the order of the time of ssh transfer and much better than the results from the most recent PIR implementation presented in [29] (note that the PIR protocol actually has weaker privacy guarantees than what we need since it does not provide database privacy), while we provide better privacy guarantees than the original SADS. In other words, we have provided strong-enough security and privacy, and at an acceptable cost.

[1] Private communication with a physician.

[2] `http://nyp.org/about/facts-statistics.html`

9. REFERENCES

[1] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Proceedings of EUROCRYPT'01*, London, UK, 2001.

[2] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of S&P'07*, Washington, DC, USA, 2007.

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

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

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

[6] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of TCC*. Springer, 2006.

[7] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13, 2000.

[8] Yan cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Proceedings of ACNS*, volume 3531, 2005.

[9] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Technical Report TR-CS0917, Dept. of Computer Science, 1997.

[10] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[11] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *EUROCRYPT*, pages 122–138, 2000.

[12] Emiliano De Cristofaro, Yanbin Lu, and Gene Tsudik. Efficient techniques for privacy-preserving sharing of sensitive information. In *TRUST*, 2011.

[13] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of CCS'06*. ACM, 2006.

[14] Emiliano De Cristofaro, Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Privacy-preserving policy-based information transfer. In *Proceedings of PETS*, 2009.

[15] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, 2005.

[16] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Sciences*, 60(3):592–629, 2000.

[17] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2004. `http://eprint.iacr.org/2003/216/`.

[18] Ian Goldberg. Improving the robustness of private information retrieval. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2007.

[19] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In *Proceedings of ICALP '08*, Berlin, Heidelberg, 2008.

[20] F. Allan Hubbell, Elizabeth B. Frye, Barbara V. Akin, and Lloyd Rucker. Routine admission laboratory testing for general medical patients. *Medical Care*, 26(6), 1988.

[21] Intel. Threading building blocks 2.2. http://www.threadingbuildingblocks.org/, 2009.

[22] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of EUROCRYPT*. Springer, 2008.

[23] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley, 1999.

[24] Brian W. Kernighan and P.J. Plauger. *The Elements of Programming Style*. McGraw-Hill, 1974.

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

[26] Michael Mitzenmacher and Salil Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 746–755, 2008.

[27] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, Philadelphia, PA, USA, 2001.

[28] Femi Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In *Proceedings of PETS*, pages 75–92, 2010.

[29] Femi Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Financial Cryptography*, 2011.

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

[31] Jitesh Shetty and Jafar Adibi. The enron email dataset database schema and brief statistical report. Technical report, USC, 2004.

[32] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *Proceedings of S&P'07*, pages 350–364, Washington, DC, USA, 2007.

[33] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of S&P'00*, Washington, DC, USA, 2000.

[34] Peter Williams and Radu Sion. Usable PIR. In *NDSS*, 2008.