

Trade-offs in Private Search

Vasilis Pappas, Mariana Raykova, Binh Vo, Steven M. Bellovin, Tal Malkin
{vpappas, mariana, binh, smb, tal}@cs.columbia.edu
Columbia University

Technical Report CUCS-022-10

Abstract

Encrypted search — performing queries on protected data — is a well researched problem. However, existing solutions have inherent inefficiency that raises questions of practicality. Here, we step back from the goal of achieving maximal privacy guarantees in an encrypted search scenario to consider efficiency as a priority. We propose a privacy framework for search that allows tuning and optimization of the trade-offs between privacy and efficiency. As an instantiation of the privacy framework we introduce a tunable search system based on the SADS scheme and provide detailed measurements demonstrating the trade-offs of the constructed system. We also analyze other existing encrypted search schemes with respect to this framework. We further propose a protocol that addresses the challenge of document content retrieval in a search setting with relaxed privacy requirements.

1 Introduction

Encrypted search — performing queries on protected data — has become increasingly interesting as concerns about security and privacy continue to grow. There are a number of variants that differ primarily in what we aim to keep private: the searchable data, queries, participant identities, etc. Schemes also differ in their expected operational environment. The majority of encrypted search mechanisms concern data outsourcing [6–9, 13, 22, 24] and to a lesser degree data sharing [10, 15, 21]. In the case of database outsourcing, one party wants to store its encrypted data on an untrusted server and be able to search it; in database sharing, one party provides a limited access to another to its database through a search protocol. This has implications for the privacy requirements. Furthermore, specific scenarios can differ in the exact results provided to a query (e.g., number of matches, document identifiers, related content, etc.), number of participants, trust assumptions, anonymity requirements, revocation of search capability and others.

All of these have profound implications for system performance. Choosing a different definition of what is “enough” privacy can make a large difference in system 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 in practice unusably costly.

In terms of privacy, intuitively the best possible result would be a protocol that yields exactly the matching results to a querier, and nothing else, as well as revealing no information to any other party. In theory, this solution could be used for all scenarios; however, this is an incomplete definition, as the intended “results” can vary, and what suffices for one situation could constitute a leak in another. Furthermore, the efficiency

This material is based on research sponsored by IARPA 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 IARPA or the U.S. Government.

requirements may differ, as well as the costs for meeting the appropriate privacy guarantees. Fortunately, privacy requirements in a scenario are often not as strict, allowing us to trade a theoretically weaker solution for better performance.

In this paper we step back from the strongest possible privacy guarantees and consider efficiency and actual requirements. These requirements include not just what may leak, but to whom; there may be other parties who are at least partially trusted. Our goal is thus not so much absolute privacy as the best possible privacy guarantees that meets the actual goals for a given scenario. Towards this end, we first present and discuss areas where we can find trade-offs between privacy and efficiency in encrypted search schemes. We define a tunable privacy framework that allows to construct a single search system that enables tuning the trade-offs at different levels, and thus can be adjusted to address different settings.

Next, we address the issue of the required output. As mentioned above (and further discussed later on), there is an important difference, both for performance and privacy, between the case where we intend to simply identify matches (or some non-sensitive function of them such as their number) and the case where we actually need to retrieve the matching document content. Accordingly, we decouple document retrieval from the actual search; we model the difference as a search system with an optional retrieval system. That, however, adds another layer to our analysis: the privacy guarantees that are made by the retrieval system. We present a privacy-preserving data retrieval protocol that can be used on top of a search system that returns document IDs.

Finally, we build and instrument a flexible encrypted search system based on the Secure Anonymous Database Search (SADS) technology suggested by [21]. It provides the user with a wide range of trade-offs, both in privacy guarantees and in document retrieval. We present performance measurements based on these trade-offs, using several different sample databases.

Paper Organization. In Section 2 we overview related work on the subject. Section 3 introduces the notions of the tunable privacy framework and search systems. We present the document retrieval protocol as a building block for such systems in Section 4. An actual example of transformation of an encrypted search scheme into a tunable privacy system and empirical evidence for the accompanying efficiency costs are given in Section 5.

2 Related Work

The setting for PIR schemes allows one party to retrieve an item at a particular position in another party's database and approaches such as [11, 14] protect the query index and do not reveal anything about the access pattern that retrieves the corresponding item without revealing any additional information about the database to the querier. This results in linear overhead in the size of the database. Chor et al. [10] generalizes the approach to allow string matching between a keyword from one party and the entries of the database of another without leaking information about the keyword and the matched string. However, their protocol requires communication linear in the length of all strings in the database. Williams et al. [24] provide an outsourcing solution that avoids touching every entry by reshuffling and reencrypting the data stored on the untrusted server in a way that prevents the server from being able to correlate the data accessed between searches.

A common technique used in encrypted search schemes [6, 22] is to use trapdoors derived from queries with the ability to determine if a ciphertext matches the trapdoor to provide search ability over ciphertexts. This capability is used to store private data at untrusted parties and execute searches on it while revealing nothing about either the query or the data. Both [6] and [22] concern the data outsourcing scenario, where a mail server is enabled to search over the encrypted emails that it stores. This method requires that the search structures produced during preprocessing be computed per searchable token (i.e. to allow keyword

search, we would need encryptions of all searchable words). This implies the search complexity will be at best linear in the number of searchable tokens. Searchable encryption can be either public key [6] (allowing anyone to add to the searchable data) or private key [22] (limiting the querier himself to adding data). Either way, only one who possesses the private key can produce trapdoors for search. In the data sharing scenario, this would necessitate a trusted party to grant search capability to others.

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 [9, 13]. Search then consists of finding the appropriate entry in the search structure for a given query’s trapdoor. This suggests trade-offs between security and efficiency based on the usage of randomized or deterministic encryption. Both [13] and [9] use deterministic trapdoors and leak the search pattern to the search server. Their search complexities, though, differ and [13] achieves linearity in the size of the matching set of document while [9] is merely linear in the number of documents. Curtmola et al. [13] also suggest a scheme that achieves security against adaptive adversaries at the price of increased storage requirements; it is linear in the number of all searchable tokens in all documents per query word stored, as well as communication complexity linear in the maximal number of tokens in a document.

Protecting the search pattern imposes efficiency costs. Bellare et al. [17] showed that in order to achieve sublinearity of the search complexity over encrypted ciphertexts, deterministic encryption is required, which leaks the search pattern. By relaxing this requirement, we can use deterministic encryption to compute tags that are efficiently searchable. One way to compute such tags is to use hashes as tags for the encryptions. However, such tags also introduce false positives in the search results due to collisions. Bloom filters [3] extend the idea of hash functions by using multiple hash functions as labels for an entry, with a parametrized false positive rate. They are used as search structures in [21] and [15] to represent the entries per document. In this way, constant search time per document is achieved. Both schemes [21] and [15] use the same term values across Bloom filters representing different documents, which allows them to improve the search time across multiple documents using Bloom filter trees [15] and bitslice storage of the filters [21]. However, this approach reveals similarity between the documents and further causes dependency between the false positive rates for different queries. The work of [21] looks for ways to mitigate the privacy leaks coming from the deterministic search structures and token by distributing limited trust to two independent intermediary parties.

Search functionality that goes beyond simple keyword match and considers conjunctive and disjunctive queries over several keywords can be achieved by providing way to generate trapdoors that handle conjunctions and disjunctions directly, which requires encryptions and the search tokens with size linear in the number of all searchable entries. A different approach has better efficiency but addresses a weaker adversarial model in which there is an intermediary party who is trusted to follow the protocol and not collude with other parties (but does not learn the data or queries).

In the case of multiple queriers in a system a revocation capability becomes an important requirement. The solution of [13] uses one key to provide all users with search capability; for each revocation, a new key is chosen by the data owner and distributed to the remaining valid users. This causes high overhead for revocations. In [21] each user has its own key, which is granted search ability via the existence of transformation keys stored at an intermediary and further provides anonymity for the querier. Revocation is then possible on a key by key basis by invalidating the transformation key, avoiding the communication overhead of distributing a new query key to everyone.

3 A Tunable Privacy Framework for Search

Encrypted search has been defined in many ways, each with its own solutions and security guarantees reflecting a different perspective (c.f., [6–10, 13, 15, 21, 22, 24]). The exact definitions of a scenario for

private search depends on the answers of the following questions:

Who provides the databases and who is the querier? — One distinguishing factor here is whether or not the data owner is the querying party. If he is, the setting is known as *data outsourcing*: one party stores its data encrypted at some untrusted server and wants to be able to retrieve search results without revealing information about the query or results. Papers addressing this setting are often based on *searchable encryption* schemes [1, 6, 22, 23], which provide the capability to generate trapdoors that allow decryption of ciphertexts that match a corresponding token, via knowledge of the secret key used to encrypt them. Encrypted search scenarios where the data owner and the querying party are different are referred to as controlled *data sharing*. PIR schemes [11, 14] provide solutions with strong security guarantees but high computation costs.

Are there any additional parties that can serve as intermediaries in the protocol? — These parties may bring utility such as computational or storage resources that are not available to the database provider or the querier. They may also be able to shoulder some of the trust, from other parties, simplifying the protection of privacy or security.

Do queries need to be authorized? — This question is relevant for data sharing. The granularity level at which search capability will be granted, e.g. keywords, groups of keywords, documents, columns in a database, has implications about how much will be leaked about the query and about the database. The coarser the search granularity, the more privacy is granted to the querier since his possible queries will be from a larger range; at the same time, though, he is given the ability to learn more about the database. At the extremes, we have the case where the querier does not need to be authorized and can submit any possible query, or where he is required to obtain a search token from some authorizing party for each query, resulting in zero privacy.

What is the search functionality and the return results? — The first question here concerns the definition of what is considered a match, whether this is an exact match between entries(keyword search), or is the result of a Boolean expression over multiple terms or the results of other queries. Once a match has been found, the next question is whether the return should be a simple handle identifying matching content, or a partial or complete retrieval of relevant content.

Does the search system have multiple users? — In the case when multiple parties are allowed to make queries to the database, protecting the identity of the querier in a specific query may be a requirement. Still, we likely desire that parties be independently authorized (or de-authorized) to make queries in the first place.

In order to capture all possible variations for a private search scenario we give the following definition that specifies the input and the output of each participant, similarly to a definition for secure multiparty computation questions:

Definition 1 *Encrypted search involves multiple parties, which fill the roles of Data Owner, Querier, or Intermediate Party. In the case of data outsourcing, data owners are also queriers. Intermediate parties, which may be required for some schemes, do not obtain any output of interest from the system as a whole. An encrypted search functionality consists of the following three phases:*

Setup: *No participant has any input. Output for each is data necessary for other phases.*

Search Preprocessing: *Participants read their own output from Setup; Data Owners read their data. Data Owners and/or some Intermediate Parties receive output which we will call search structures.*

Search: *Participants read their own output from the Setup; those with search structures read this as well, while Queriers read queries. Queriers output search results for their queries.*

The strongest notion of privacy for the above protocol requires that the output of the queriers from the search protocol are exactly the matches for their queries, and no other party learns anything. In the interest

of efficiency, weaker privacy guarantees may be sufficient; it is often acceptable to leak some controlled amount of information, such as:

- **Information about the queries:** Parties other than the querier may learn about a query, or multiple queries, or correlations among them.

- *Leakage from encrypted query* — Deterministic encryption is one example known to leak certain types of information, but as shown by Bellare et al. [17] is a necessity to achieve better than linear complexity in the number of searchable entries. While the implications for a single query will depend on knowledge of the encryption key by the other parties, for multiple queries this can lead to leakage of the search patterns, such as when the same query is submitted more than once, perhaps by different users.
- *Leakage from results* — The implications of such leakage depend on the party who learns the matching results and what exactly constituted a result. If the data owner learns a pointer to the matching content (document, record) or the actual return result includes the content and it is leaked to an intermediary party, the exact matching content may allow inference about the submitted query. Another possibility is pattern leakage; a party may learn only pointers to matching content such as document ids but be able to detect overlap between the return results for different queries.

- **Information about the database:**

- *Leakage from the return results* — A system with a false positive rate may return excess results beyond what is appropriate for a query. This is a privacy concern in the case of data sharing when the querier is not the owner of the database and should not be able to view documents besides correct results. Conversely, in data outsourcing this would not constitute a privacy leak since the querier owns the database and can access its whole content. If this type of leak is equivalent to a false positive for the matching functionality, it is natural to ask about the case of a false negative (i.e. not all matching records are returned). While it is not a privacy leakage it affect the correctness of the output.
- *Leakage from search structures:* Here the search structures computed in the preprocessing stage may leak information about the database such as similarity of data records. This is a privacy concern when the search structures are given to a party different from the owner which may be the case if we want an intermediary party to execute the search in order to prevent search pattern or result pattern leakage to the data owner.

As we have discussed above, there are many different privacy and security settings that can be selected for a given search scenario, given its security requirements, its efficiency requirements, and the available resources. If we want to achieve a search system applicable to a range of different scenarios, we should provide controls that would allow the user to tune the system at the appropriate privacy and efficiency levels for his needs. We can do that by having a modular system with blocks of system functionality that can be instantiated in several different ways according to our needs. We define a practical system as follows:

Definition 2 *A tunable privacy search system is a one that implements a protocol such that:*

- *The protocol is partitioned into independent modules.*
- *The functionality of at least one of the modules can be instantiated with more than one implementation.*
- *The search schemes resulting from the different instantiations of the modules provide different privacy guarantees and costs.*

The flexibility of a tunable privacy search system will depend on the granularity of the protocol partitioning and the different instantiations for each part, which in turn will depend on the approach of the search scheme. In Section 5 we show how to transform the SADS scheme of [21] in such a system.

4 Document Retrieval

As we discussed in Section 3, the goal of a search protocol can either be to identify of the data of interest to the querying party (output document IDs) or to retrieve the contents of the matching documents. Document retrieval poses its own challenge, especially when the querier and the data owner are different parties. 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 querier. Thus, a protocol that meets our strongest security Definition 1 would need to touch the contents of the entire database [10]. This leads to a large increase in both the computation and communication complexity compared to just returning document IDs. One way to avoid this prohibitive cost is to relax our security definition and allow leakage of the search result pattern. 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 [6, 9, 13]. This approach, however, is not applicable to the case of data sharing, where leaking the matching documents to the data owner reveals more than the result pattern: the owner also knows the content of the documents, from which he can infer information about the query.

We address this by constructing a document retrieval scheme that can be used on top of any other scheme that returns document IDs. We maintain 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 obviously without learning the search results. 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), which utilizes encryption with the following property (defined in more detail in [21]):

Definition 3 (Encryption Group Property) Let $\Pi = (\text{GEN}, \text{ENC}, \text{DEC})$ be a private key encryption scheme. We say that Π has a group property if $\text{ENC}_{k_1}(\text{ENC}_{k_2}(m)) = \text{ENC}_{k_1 \cdot k_2}(m)$ holds for any keys k_1, k_2 and any message m .

Protocol for Document Retrieval

Storage Reencryption (performed during the search *preprocessing* phase)

Inputs: $S \leftarrow$ documents D_1, \dots, D_n , keys k_1, \dots, k_n and k'_1, \dots, k'_n ;

$P \leftarrow$ permutation π of length n ;

$S, P \leftarrow (\overline{GEN}, \overline{ENC}, \overline{DEC})$, an encryption scheme meeting Definition 3

Outputs: $S \leftarrow \perp$; $P \leftarrow \overline{ENC}_{k'_{\pi(i)}}(D_i)$ for $1 \leq i \leq n$

1. S sends to P $c_i = \overline{ENC}_{k_i}(D_i)$ for $1 \leq i \leq n$.

2. For $1 \leq i \leq n$ S and P execute oblivious transfer protocols that allow P to obtain $k'' = k_i^{-1} \cdot k'_{\pi(i)}$.

3. For $1 \leq i \leq n$ P computes $\overline{ENC}_{k''}(c_i) = \overline{ENC}_{k_i^{-1} \cdot k'_{\pi(i)}}(\overline{ENC}_{k_i}(D_i)) = \overline{ENC}_{k'_{\pi(i)}}(D_i)$.

Document Retrieval (performed at the end of the search phase)

Inputs: $S \leftarrow$ keys k'_1, \dots, k'_n ;

$P \leftarrow$ permutation π of length n , $\overline{ENC}_{k'_{\pi(i)}}(D_i)$ for $1 \leq i \leq n$;

$C \leftarrow$ query Q ;

$S, P, C \leftarrow$ encrypted search scheme $EncSearch$ that returns IDs of matched documents to P, C .

Outputs: $S \leftarrow$ cardinality of the output set from $EncSearch$ for query Q ;

$P \leftarrow$ IDs of documents matching query Q from $EncSearch$;

$C \leftarrow$ the content of the documents matching Q from $EncSearch$.

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

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 $k'_{\pi(i)}$ 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 4 of the re-encryption can be implemented using protocols for oblivious transfer [2, 12, 19].

5 SADS as a Tunable Private Search System

We now extend the SADS scheme for encrypted search [21] into a tunable search system by identifying parts that permit different implementations with tradeoffs between privacy guarantees and efficiency. These controls provide choices between functionality, architecture, and underlying algorithms. Such flexibility allows the search system to be mapped to the specific requirements of various settings.

5.1 Secure Anonymous Database Search (SADS)

We start with a brief overview of the SADS scheme; for further detail refer to [21]. SADS offers a search protocol between a data owner/server (S) and a search client (C) with the help of two neutral intermediary parties, an *Index Server* (IS) and *Query Router* (QR), involving the following three protocols:

Re-routable Encryption. This protocol allows for a party that is responsible for routing messages between senders and receivers. This party is trusted to match senders and receivers, but not to learn the messages. Optionally, the party may be required to transform the messages before forwarding.

PH-DSAEP+. PH-SAEP+ is a private key encryption scheme which combines the Pohlig-Hellman function [20] and the SAEP+ (short for Simple-OAEP) padding construction introduced in [5]. This encryption scheme has the group property (as in Definition 3), which is necessary to implement re-routable encryption. PH-DSAEP+ is a deterministic variant of PH-SAEP+.

Bloom Filters. Bloom filters [4] allow efficient testing of set membership. A Bloom filter (BF) is an m -bit array B . Initially, all bits are set to zero. There are n independent hash functions H_i , $1 \leq i \leq n$, with outputs in the range $[0, m - 1]$. To add an entry W to the Bloom filter, we calculate the following values:

$b_1 = H_1(W), b_2 = H_2(W), \dots, b_n = H_n(W)$ and set $B[b_i] = 1$ for each $1 \leq i \leq n$. To check whether W is present in the filter, we compute the indexes b_1, \dots, b_n as above. If $B[b_i] = 1$ for all $1 \leq i \leq n$, the entry is present; if any of the bits are 0, W is not present. In the SADS scheme blocks from the deterministic PH-DSAEP+ encryption ciphertexts of the entries are used as the corresponding hash values for the Bloom filters.

SADS Construction. (Figure 1) Given a server(S), a client (C), a query router (QR), and an index server (IS), a secure anonymous database search (SADS) scheme is defined 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 result return. To authorize C to search S, QR and C run a key exchange protocol twice, producing a ratio key for QR between S’s encryption key and C’s query submission key, and another ratio key between IS’s encryption key and C’s result return 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:** The query is delivered using the following re-routable encryption protocol: 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 also returned using re-routable encryption: IS encrypts R with PH-SAEP+ and sends it to QR; QR re-encrypts the ciphertext with the result return transformation key for C, and sends it to the client.

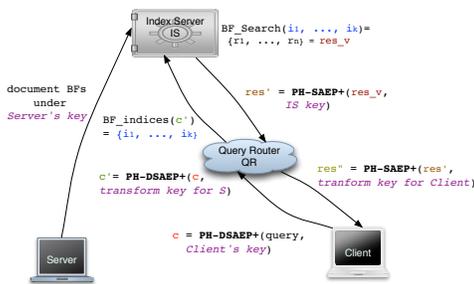


Figure 1: SADS Query.

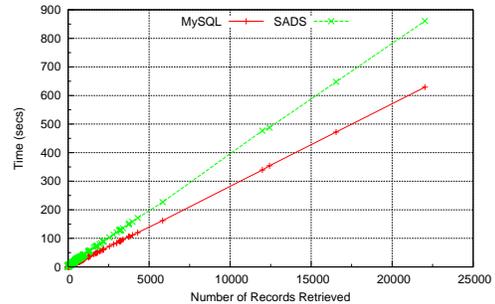


Figure 2: Comparison between our scheme and MySQL.

5.2 Tunable Controls

We now identify potential tuning switches that can vary privacy and efficiency trade-offs.

Search Functionality. SADS returns to the user identifiers for matched documents. This scheme can optionally be extended with the document retrieval protocol from Section 4 to additionally deliver content for matching documents. We then have the following choices:

- If the client directly requests decryption keys from the data owner, it leaks information about the pattern of results over multiple queries. To avoid this the client may request the keys for them all at once or in subsets over different times. Furthermore, he can avoid requesting multiple times a key that repeats in multiple queries. However, since complete results need to be signed by the IS, this requires additional per-key signatures.

- As we discussed in Section 4 encryption schemes with the group property are generally not efficient. We can instead use a more efficient encryption scheme for long documents and then apply encryption with the group property to protect the encryption keys used for the documents as described in Section 4. An obvious implementation is to encrypt the documents with AES and then use PH-SAEP+ for encryption of the AES keys. A trade-off here may be connected to the security guarantees provided by the two encryption schemes.

Search Architecture. The architecture suggested by the SADS scheme distributes trust to the intermediary parties. If the existence of two non-colluding intermediary parties is not viable for a particular scenario, and client anonymity is not required (for example, there is only one client), we can remove the QR. Client authentication and revocation, normally enforced by the QR, can be guaranteed by other means such as separate authentication protocols, revocation lists, or regular updates of all client keys.

Preprocessing. During the preprocessing stage, a Bloom filter is generated per document containing all keywords. In the SADS scheme, adding a keyword to a document involves encryption of the keyword under the key of the server. 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 will avoid some recomputation, but requires additional storage space. Whether to do this, and how much to cache, will depend on the nature of the documents and repeat frequency. This is also applicable if the preprocessing of a keyword is not identical but shares a common intermediary result that can be reused, which is the case if we use different hash function sets per document as we discuss next.

Bloom Filters Over Multiple Documents. The Bloom filters of different documents in SADS share the same hash functions, and 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, the expected number of false positive results in a database with N documents will be $FP_{single} \cdot N$. 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 filter 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.

5.3 Trade-off Measurements

We now analyze a number of different trade-offs for the SADS scheme, which is roughly 4 Klocs of C++. We measured both the preprocessing and the search phases. More precisely, we measured the performance gain for various workloads when we (i) cached some computation results and (ii) parallelized the preprocessing phase. In addition, we quantify the trade-offs between privacy and performance in the search phase by computing the average query time over a set of 100 queries, while changing the dataset size. We repeated that for different configurations using the switches previously described.

Datasets used. We used two distinct datasets of different type: a large collection of text documents from the Enron Database (as in [21]) and a 51-column CSV (Comma Separated Values) file with random data. In

the case of the Enron database we created a Bloom filter per email with stemmed words as the entries; for the CSV data we created a Bloom filter per record with entries of the form "column name || column value".

Hardware and evaluation setup. Our measurement platform was two servers and a 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.

5.3.1 Memory consumption

Along with the timing measurements, we also monitored the memory consumption of our 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 a 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 query router used a small, constant amount of memory (~2MB). On the other hand, the index server'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 them (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.

5.3.2 Caching exponentiation values

The first experiment measured the impact of caching the exponentiation values during the preprocessing phase. The caching capability we added uses LRU removal policy.

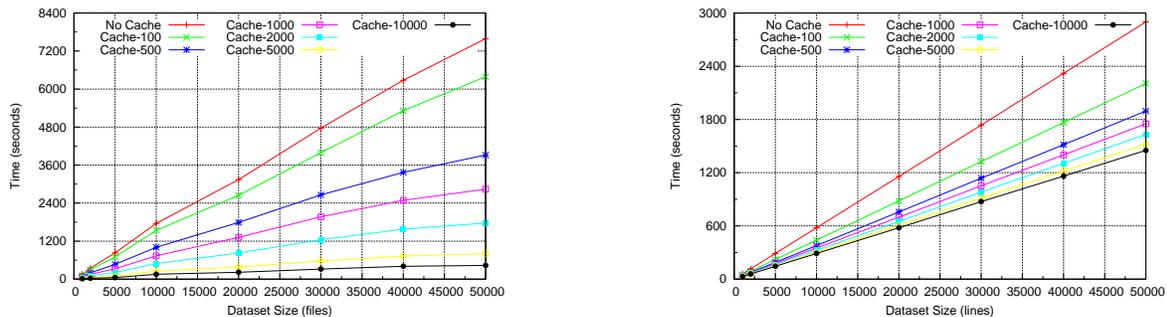


Figure 3: Duration of the preprocessing phase for text files (left) and CSV data (right).

Figure 3 shows the caching effect on the preprocessing phase for both the text and CSV datasets. The x axis is the dataset size and the y axis is the preprocessing time in seconds. It is clear from these figures that the exponentiation values cache has a greater impact on the text files than the CSV data; this is a result of the greater randomness of the CSV data. Figure 4 shows the cache hit ratio for the same cases.

5.3.3 Parallel preprocessing optimization

Multicore architectures are dominant in today's computers. To take full advantage of such architectures, one has to "parallelize" applications so they can run simultaneously on all available cores. For SADS preprocessing, this was straightforward. As stated earlier, SADS processes each item independently, and

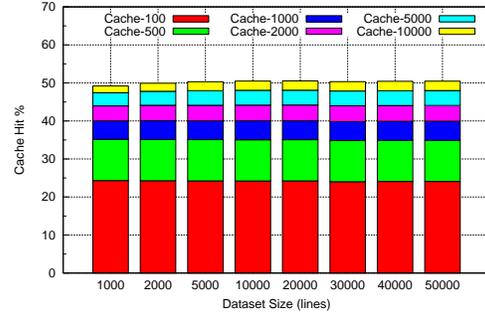
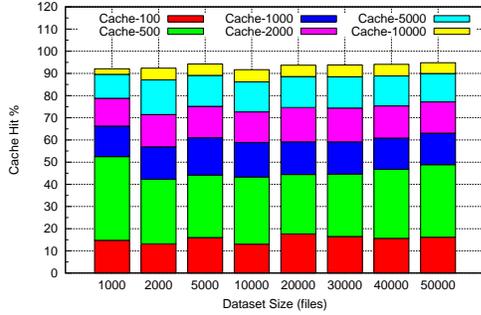


Figure 4: Cache hit ratio on exponentiation values during preprocessing phase for text files (left) and CSV data (right).

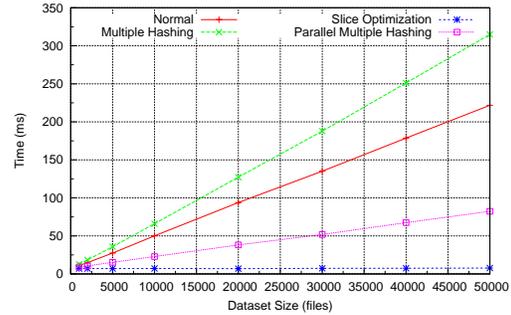
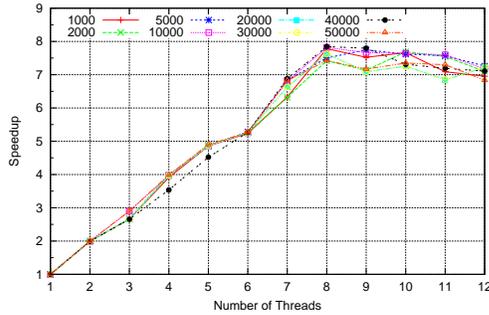


Figure 5: Speedup in parallel preprocessing on an 8-core server.

Figure 6: Average query time of the normal case, the slicing optimization, the multiple hashing scheme and the parallelized multiple hashing scheme (using 8 threads on an 8-core server).

furthermore handles each element of each item separately. For example, an item can be a file and an element can be a word; alternatively, an item can be a line in a CSV file and an element can be a value in that line. For each item being processed, a distinct Bloom filter is created and the item’s elements are inserted after the cryptographic transformations take place. This computational independence makes for simple and robust parallelization.

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. We used the open source Threading Building Blocks library [16]. It is easy to use and well-integrated with C++. It took roughly 10 lines of code to parallelize the entire preprocessing phase for the CSV data handler.

Figure 5 shows the speedup we gained from parallelizing the preprocessing phase while executing it on our 8-core servers using different sizes of datasets. The speedup grew with the number of threads, maxing at 8, the number of the cores of the server. After that point, there were no further gains, and in fact a minor loss due to thread scheduling overhead.

In addition to parallelizing the preprocessing phase, we also implemented a parallel version of the Search phase when using multiple hash functions, as described below.

5.3.4 Same vs Multiple hash functions

The next trade-off we considered is the Bloom filter implementation. There are two options; we can either use the same hash functions among all the documents in the dataset, with expected performance gains and the use of bit-slicing optimizations, or we can use different hash functions for improve privacy guarantees.

In practice, rather than using distinct hash functions, we key our has function off of the document

identifiers. We additionally need to support a number of different hash functions per document. We implemented that by using the technique presented in [18] for generating a group of hash functions using a family of 2-universal hash functions. In our implementation, we used HMAC using MD5 and SHA1 (using the document’s ID as their 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) and $H_2(w)$ is HMAC(MD5, ID, w).

Figure 6 shows the comparison for the above three schemes: normal with slicing optimization, with multiple hash functions, and with the parallelized version of multiple hash functions. As expected, the average query time grows linearly in the “Normal” case, as the actual search is done linearly over all the Bloom filters. Next, we can see that the slicing optimization is of great value as it keeps the average query time almost constant over the corpus size. Using the multiple hashing scheme, we do get better privacy guarantees, but with the cost of increasing the Normal case by a another factor that is proportional to the corpus size. That is because for each document we want to search, we have to recalculate the hash functions. 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 6.

In conclusion, the cost of the multiple hashing is twofold. Firstly, it makes the slicing optimization unusable; second, it adds the additional overhead of recalculating the hash functions per document.

5.3.5 Boolean operations on query terms

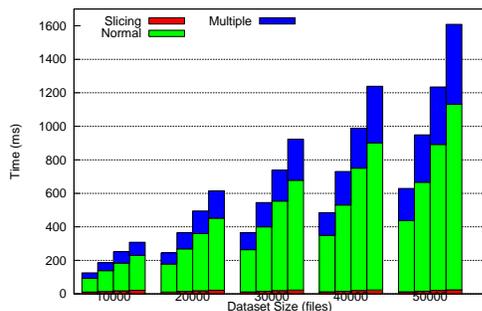


Figure 7: Average query time of the normal case, the slicing optimization and the multiple hashing scheme. Each cluster is for a different dataset size and each bar is for a different term count (from 2 to 5).

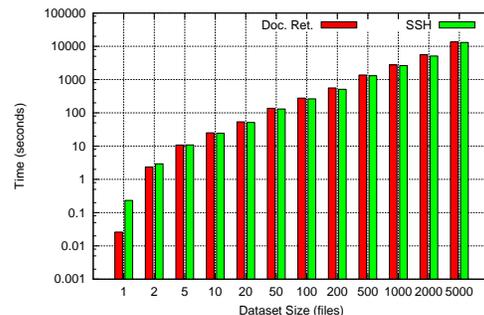


Figure 8: Average time for retrieving documents anonymously, compared to retrieving them non-anonymously using ssh file transfer. Average size of files being transferred was 27.8 Mb

SADS also supports Boolean Queries. One naive way to do this is to compute term queries separately and union or intersect the results. However, Bloom Filters can more efficiently handle ANDs by combining indices into a superset, and ORs can be handled in parallel when using a bit-slicing optimization as described in SADS¹. In the normal case, where we search linearly over the documents, the optimization we implemented was to skip documents that already contained one of the terms. That way we avoided searching over such documents, thus reducing the overall querying time. In the case where the slicing optimization was enabled, we run each of the ORed queries in parallel. This approach has two main benefits. First, it has better cache behavior because we fetch each slice once and use it for all the result vectors; second, in some cases it permits us to avoid reading several slice portions if the corresponding bits of all the result vectors have been zeroed out.

¹As mentioned in [21]

Figure 7 shows the results of timing Boolean OR queries of variable terms. The y axis is the query time in milliseconds. Each cluster of bars is for a different dataset size; each bar is for a different term count. The first bar is for two terms, the second for three, and the last two for four and five, respectively. As we expected, the best configuration as far as the performance is concerned is when the slicing is enabled; these are the red parts that can be hardly distinguished on the bottom of the bars. We see that the overall time is again almost constant across the different dataset sizes (as on Figure 6), but it does increase with respect to the number of terms. The same thing is also true for the normal and the multiple hashing case. In both of these cases we see that the search time increases with the number of terms, and of course with the dataset size. The performance gain we have in all cases is sub-linear in the number of query terms.

5.4 Document Retrieval

As mentioned earlier, a parameter in functionality of the search system is whether it returns IDs of matching documents or the actual documents themselves. We implemented this as a privacy-preserving document-retrieval add-on module.

We implemented document retrieval using PH-SAEP and using 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 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, \dots, 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 8 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.

5.5 Overall Performance

In this section we compare the performance of our scheme with a real world DBMS, MySQL. In order to do that, we implemented a SQL front-end for our system that could parse simple conjunctive and disjunctive queries. Then, we loaded the CSV dataset to both systems and we executed a number of queries of variable resultset size. During that evaluation, SADS was configured to use multiple hash functions and document retrieval was enabled, so as to be able to retrieve the actual data. Figure 2 shows the total duration of both the query and the retrieval of the data for SADS and MySQL. As show, our scheme performs just 30% worse on average than MySQL, where on the same time it provides privacy.

6 Conclusion

When we consider the question of encrypted search in practical settings, the privacy guarantees of a scheme are no longer the only relevant issue: a perfectly secure scheme that is too inefficient will see no use. The efficiency of an approach becomes a major factor in determining its usability given the available resources. Schemes that leak nothing beyond the exact matching results to the querying party are often too expensive to be usable. Furthermore, in many cases looser privacy guarantees suffice; such a relaxation of the security

definitions allows us to achieve much better efficiency, which in turn makes the scheme a viable solution for a much wider range of problems. The most appropriate trade-offs between privacy and efficiency are different in different settings. We have proposed a privacy framework based on the SADS [21] search protocol that makes a step in the direction of relaxing security guarantees to gain efficiency by using deterministic encryption for search structures and achieving better than linear search complexity in the number of searchable tokens. Our framework provides the following privacy setting options with their corresponding efficiency trade-offs:

- Document search providing document indices as return results without leaking any information to the document owner. The search is executed by the intermediary party (IS) that receives search structures from the owner.
- Document retrieval of the search result documents leaking only the number of accessed documents to the owner. Data owner and IS have to execute a document re-encryption protocol during the preprocessing phase.
- Access control and querier anonymity when the system is used by multiple users. A query router (QR) transforms and routes the queries.
- Encryption options for the database documents: using expensive transformable public key encryption for the whole documents, or using weaker AES encryption for the documents and encrypting only the keys with the more expensive scheme.
- Slicing techniques for search structure storage which improve performance, or less efficient structures that protect against document similarity leakage.

Furthermore, we enable caching and multithreading optimizations that improve the efficiency of the system for the appropriate data and hardware. Our implementation consists of well defined modules handling cryptography functions, search structure storage, and data handlers that facilitate easy extension of the system.

References

- [1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. *J. Cryptol.*, 21(3):350–391, 2008.
- [2] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 119–135, London, UK, 2001. Springer-Verlag.
- [3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [4] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
- [5] Dan Boneh. Simplified OAEP for the RSA and Rabin functions. *Lecture Notes in Computer Science*, 2139:275–291, 2001.
- [6] 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.
- [7] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows PIR queries. In *Proceedings of CRYPTO'07*, 2007.

- [8] 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.
- [9] Yan cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, volume 3531, 2005.
- [10] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Technical Report TR-CS0917, Dept. of Computer Science, Technion, 1997.
- [11] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
- [12] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *EUROCRYPT*, pages 122–138, 2000.
- [13] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88, New York, NY, USA, 2006. ACM.
- [14] 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.
- [15] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2004. <http://eprint.iacr.org/2003/216/>.
- [16] Intel. Threading building blocks 2.2. <http://www.threadingbuildingblocks.org/>, 2009.
- [17] A. Boldyareva M. Bellare and A. O’Neill. Deterministic and efficiently searchable encryption. In *Proceedings of CRYPTO’07*, 2007.
- [18] Michael Mitzenmacher and Salil Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. In *In Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 746–755, 2008.
- [19] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA '01: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
- [20] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.
- [21] Mariana Raykova, Binh Vo, Steven Bellovin, and Tal Malkin. Secure anonymous database search. In *CCSW 2009.*, 2009.
- [22] 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.
- [23] B. Waters, D. Balfanz, G. Durfee, and D. Smetters. Building an encrypted and searchable audit log. In *NDSS 2004.*, 2004.
- [24] Peter Williams and Radu Sion. Usable PIR. In *NDSS*, 2008.