# Secure Anonymous Database Search

Mariana Raykova
Columbia University
New York, USA
mariana@columbia.edu

Binh Vo
Columbia University
New York, USA
binh@columbia.edu

Steven M. Bellovin
Columbia University
New York, USA
smb@columbia.edu

Tal Malkin
Columbia University
New York, USA
tal@columbia.edu

## ABSTRACT

There exist many large collections of private data that must be protected on behalf of the entities that hold them or the clients they serve. However, there are also often many legitimate reasons for sharing that data in a controlled manner. How can two parties decide to share data without prior knowledge of what data they have? For example, two intelligence agencies might be willing to cooperate by sharing documents about a specific case, and need a way of determining which documents might be of interest to each other.

We introduce and address the problem of allowing such entities to search each other's data securely and anonymously. We aim to protect the content of the queries, as well as the content of documents unrelated to those queries, while concealing the identity of the participants. Although there exist systems for solving similar problems, to our knowledge we are the first to address this specific need and also the first to present a secure anonymous search system that is practical for real-time querying. In order to achieve this in an efficient manner, we make use of Bloom filters [5], definitions of security for deterministic encryption [22] that we adapt and instantiate in the private key setting and of a novel encryption primitive, reroutable encryption.

## Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; E.3 [**Data**]: Data Encryption

## General Terms

Security

## Keywords

encrypted Bloom filters, private information retrieval, anonymity, database, deterministic encryption, encrypted search

## 1. INTRODUCTION

Often, different parties possess data of mutual interest. They might wish to share portions of this data for collaborative work, but consider the leak of unrelated portions to be a privacy issue for themselves or their clients. Thus, methods that provide a well-defined and secure sharing of the data between untrusting parties can be useful tools. One such method that we introduce in this paper, is the ability for a client to search the information residing on another server without revealing to the server his identity or the content of his query; at the same time, it is desirable to guarantee that query capability is only granted to appropriate clients and that they do not learn anything unrelated to the query. Such a tool is useful in deciding and agreeing upon information-sharing between parties who do not initially know if they have data worth sharing with each other, and do not want to share information until they do. In addition the very fact that a client is interested in running certain queries is considered sensitive, and thus both his identity and the query content must be protected from the server.

The system we are proposing has many possible applications. For example, two intelligence agencies might like to search each other's data to discover if they have complementary information about the same parties. Similarly, the police may need to search the databases of different institutions, e.g., banks for information about people suspected of embezzlement. Even outside of law enforcement, this type of search might be useful to a physician who wants to find out about other patients with the same rare disease as a patient of his own, along with treatment methods that have given good results. Or institutions might wish to protect logs containing sensitive information about the activities of their members, and yet allow restricted searches on information about suspicious behavior that, when correlated across different domains, may help detect attacks. These scenarios all present a common problem: a facility has data that legitimately could or should be shared with another party, embedded within a large amount of data that should be held


*CCSW'09,* November 13, 2009, Chicago, Illinois, USA.


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.

confidential. Further in cases such as business acquisition research and law enforcement the identity of the querier needs to be kept anonymous to avoid causing fluctuation in share prices or tipping off investigation.

**Our Contributions.** We address the above concerns by defining and implementing Secure Anonymous Database Search (SADS). Although the framework can support more general queries, we focus here on the specific functionality of keyword search, which allows an authorized client to anonymously and securely query a server for documents containing a desired keyword. We design an efficient SADS scheme, and provide for it proofs of security and performance evaluation.

Although it sounds similar to existing work on encrypted search, our system differs in a significant manner: existing papers assume that the data being searched is owned by the querier and must be protected from the server [8,9,11,14,22, 26]. In our scenario, the querier and the owner of the data are different parties. This constitutes a different adversarial model: we must protect the database from the querier. Additionally, we aim to protect the identity of the querier which also introduces new issues, such as how to ensure that the data owner can prevent arbitrary unauthorized parties from sending queries. The PIR/SPIR line of work [13, 16] assumes a similar scenario to ours in that an authorized user is requesting some data from a server who owns it, but differs in that they do not aim to protect the identity of the user, and also (with a small number of exceptions, e.g., [12]). merely support simple selection, not full query capability.

Another important distinction from existing work is that we aim to achieve time efficiency that is sufficient for real-time search, something unseen in both encrypted search and PIR solutions. As in [22], we consider this to mean sublinear search time in the total size of the searchable data. Protocols such as [8, 11, 26, 27] achieve linear search time; to improve this complexity, we may be willing to sacrifice strict definitions of privacy and security in a limited and measurable manner. Thus, our goal is to guarantee practical performance and achieve the maximum privacy, security and anonymity possible under the efficiency requirement.

**Our Approach.** We introduce two parties, the query router (QR) and the index server (IS), which will serve as intermediaries in the search protocol. They are trusted with limited information necessary to perform their functions in the scheme. The index server stores encrypted search structures produced by the data owner and executes submitted queries without learning the query content or anything about the underlying database. The query router connects queriers to index servers without revealing identities of either participant to any other entities. He further translates encryption by the client to encryptions appplicable to the IS index ensuring that the results cannot be traced. To accomplish this, we introduce a new primitive that we call reroutable encryption, which we use for the query submission and result return via the query router to the index server. To achieve efficiency of the search algorithm we use Bloom filters [5] as our search structures. We use the security model introduced in [22] that allows sublinearity of search schemes. We adapt the definition for IND-CCA secure deterministic encryption from [22] for use in the private key setting and use this scheme to produce ciphertexts from which we build our Bloom filter search structures.

**Organization.** In Section 3 we more formally describe the problem we are addressing and our solution approach. Section 4 describes the building block protocols that we use for our system. We present our complete scheme with its security properties in Section 5. The implementation structures used to store Bloom filters and optimize parallel search across multiple filters are discusses in Section 6. We follow with performance measurements that demonstrate this system is efficient and practical in Section 7.

## 2. RELATED WORK

The problem of secure anonymous database search can be considered in general as a special case of the problem of secure multiparty computation [18, 30, 31] and be solved with general techniques such as [18]. Although this approach provides strong privacy guarantees for the participating parties, the time complexity of generic secure multiparty computation schemes is too high to be practical for real world use.

Protocols for Private Information Retrieval (PIR) [13] and Symmetric Private Information Retrieval (SPIR) [16] provide a limited type of privacy preserving search. The scenario that PIR addresses is between two parties: server and user, where the server has a database of $n$ items and the user wants to obtain the item at position $i$ without the server learning the value of $i$. In the case of SPIR, it is additionally required that the user does not learn any other item except the one that was requested. These protocols have sub-linear communication and polynomial computational complexity, which may be better than a generic multiparty protocol, but are still inefficient for practical uses. Additionally, these protocols typically support only simple selection, rather than general query capability (a notable exception being [12]).

Papers that present encrypted search address a different type of search scenario from ours, namely database outsourcing [1,8–11,14,26,28,29]. In this setting one party possesses data but does not have enough resources to store it. He keeps the data on an untrusted storage server, but mainains the ability to search the data without leaking any information to the server. As in our search scenario, the query must be protected from the server, however in our system the data is owned by the server. Since in these systems, each user's data is encrypted and only readable by themselves, protecting anonymity and handling authentication are relatively trivial matters.

Many approaches to this problem use encryption systems that allow matching of ciphertexts of the same encrypted word [8–10,26]. However, this approach only aims to protect the query content under the assumption the querier owns the data; the server could otherwise easily match the queries to its own data text. Most developments in improving privacy and efficiency also rely on this assumption: [14] suggests the querier preprocess the data by computing inverted indexes on search words, and [29] suggests the user reorder the data on the storage server between queries to prevent the server from learning access pattern.

Another common technique is to grant search capability for a keyword by providing a trapdoor that allows decryption only of encryptions of that keyword [8–10, 26]. This, however, requires computational complexity linear in the number of words stored at the database. As with [22] we consider this running time unacceptable, and aim to achieve performance comparable to non-private search. Linear time is a trivial search time that has proven to be extremely inefficient for large data sets. Bellare et al. [22] show that in order to achieve better than linear complexity of search

the underlying encryption scheme must be deterministic, which presents a clear tradeoff between efficiency and the strong privacy guarantees that come with non-deterministic encryption.

Papers like [27] and [25] present the scenario most similar to ours: the problem of log sharing in which investigator obtains from an authorization party the capability to search encrypted audit logs for particular words or values. To accomplish this, queriers are given all encrypted keywords in the documents, but are ensured to be only capable of decrypting matching ciphertexts and consequently incurring linear computational cost. Since each querier must receive all of the encrypted text, the communication complexity is not constant. If the encrypted logs are stored and matched by a third party instead, we run the risk that a possible leak of the encryption key will reveal all sensitive data.

Our approach addresses the issues of efficiency and database protection using Bloom filters as a basis for our search structures. Bloom filters can be much smaller than the data they represent and cannot be used to reconstruct the original data. The works of [3] and [17] also utilize Bloom filters for search purposes. However, [3] uses weaker privacy definitions. The scheme of [17] combines Bloom filters with one way functions to achieve security guarantees but addresses a different setting where the trapdoor giving search capability for a keyword has to be generated by the owner of the private key used for the generation of the index. This is not applicable for our scenario since we would like to hide the search term from the database owner and search capabilities are granted for a collection of documents as opposed to separate words.

One of the problems we address is limiting permission for search capability to sets of users. In schemes like [8, 26, 27], permissions are granted on a per-term basis by including a trapdoor allowing decryption of ciphertexts of this particular word. However, this leaks information about what queries parties will make to the authenticating authority, since they must request trapdoors for query terms. We instead allow setting different granularities of search capability, where one can allow search on the whole database, on particular documents or if some words are especially sensitive one can further restrict the search capability for those.

A final major addition we provide over existing schemes is the protection the querier's identity from the server. An important tool we use to provide this is re-routable encryption. Similar to this tool are the ideas of proxy encryption [4] and universal re-encryption [21]. Proxy encryption [4] allows one party to decrypt on behalf of another party. For this purpose the two parties compute and publish a transformation key between their encryptions that allows an untrusted proxy to convert ciphertexts for the first party to ciphertexts for the second party without learning anything about the key of any of them. Our re-routable encryption instead assumes that only the intermediary part will know the transformation key, and will use this to protect identities of the participants from each other. Universal re-encryption [21] allows a party given a ciphertext to obtain a different encryption of the hidden message under the same key but with different randomness without learning the message. However, it does not provide for transformations between encryption under different keys.

# 3. SECURITY ARCHITECTURE

## 3.1 Problem Setting and Requirements

The general scenario we consider includes multiple parties who possess private sensitive data, which they are willing to share under certain very specific circumstances. Each execution of the scheme will involve a party who owns a set of documents he wishes to make available for secure anonymous keyword search by authorized parties. Any other party may be authorized to take the role of the querier, whose input is some keyword that he wishes to search for in the database. We will interchangeably refer to the first party as data owner or server, and the second party as querier or client. The goal is for the protocol to meet the following requirements:
*Correctness:* The querier's output consists of all the matches, namely the indices of all documents containing the keyword. A tolerated probability of expected error (false positives or negatives) may be specified.
*Client Security:* The data owner does not learn any information about the query (keyword).
*Server Security:* The querier learns nothing about the data except for the specified output (matches) for his query.
*Server Access Control:* Only parties authorized by the data owner can submit queries and receive outputs for this data.
*Client Anonymity:* The data owner learns no information about the identity of the querier as chosen from amongst the pool of authorized parties. This also precludes information about linkage of two queries coming from the same client.

We note that some application scenarios may require also *server anonymity*, and our system can easily support this (using the same mechanisms we use for client anonymity).

The formal definitions of the above properties (e.g., what "learns no information" means) can be specified following standard cryptographic notions.

Finally, we emphasize that *practical efficiency* is quite a central requirement for our system, and we design our model and protocol accordingly. In particular, high communication complexity, or per-query computation complexity that scales linearly with the number of words in a document will not be acceptable. This rules out the use of existing generic cryptographic techniques from secure multiparty computation [18, 30, 31] and PIR [13, 16].

## 3.2 The SADS Model and Protocol Structure

It is not hard to show that the security, anonymity, and efficiency requirements stipulated above are conflicting, and cannot all be achieved simultaneously without adjusting the model. For example, sublinear computation and constant communication conflict with client privacy, as they allow the server to gain information on the answers to the query, and thus on the query itself. Client anonymity seems to conflict with server access control, and obviously anonymity cannot be achieved if the server and client are the only two parties participating in the interaction. Trying to solve the latter problem by involving all parties in the system for each search is not practical (both in terms of efficiency, and since it requires a fixed and known set of parties).

Instead, we will expand our model by adding two new parties that will participate in each search, the *Index Server (IS)* and the *Query Router (QR)*. These may be viewed as neutral parties available to regulate the data sharing process without learning the participants' private inputs. The security and anonymity requirements with respect to these new par-

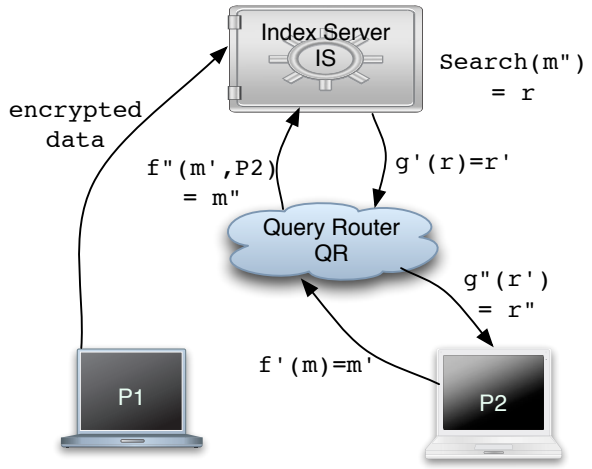



**Figure 1: General Setup: P1 makes its data available for search providing IS with search structures, P2 submits keyword queries anonymously to IS via QR, IS sends back the search result though QR**

ties will be reasonable, but weaker than those between the client and server; in return, they allow us to achieve practical efficiency. Before elaborating on these requirements (which will complete our definition of SADS), we overview the general architecture of our SADS protocol, demonstrating the roles of IS and QR and their trust implications.

Figure 1 illustrates the search protocol. A database owner (P1) generates a search structure computed from (an encryption of) his data and gives it to the index server. This structure enables IS to answer (encrypted) queries but does not reveal information about the provided database. Outsourcing the search to IS prevents the data owner from finding out the results to encrypted queries. The IS sees the results, but does not know what documents they correspond to. At most, the IS will be able to tell when two submitted queries have overlapping results. This is mitigated by preserving the anonymity of the queriers with respect to the index server. However, providing such anonymity introduces a new problem: how to guarantee that only authorized users are submitting queries. This is addressed by the query router, who serves as an intermediary in the communication path between querier and IS. QR is trusted to know and protect the identities of the participants, while enforcing correct authorization before allowing queries to reach the IS. However, he is not trusted to see the content of the queries or results. Thus a querier (P2 in Fig. 1 ) will submit his encrypted query to the QR, who checks the authorization of the user, transforms the query and forwards it to the IS. The IS will send back search results to the QR, which will be able to forward them to the respective user. The results are encrypted so that the QR does not learn anything.

With this architecture in mind, we make the following requirements with respect to IS and QR

*Data Security Against IS and QR:* Both IS and QR learn no information about the data.

*Client Anonymity Against IS:* IS learns no information about the identity of the querier. This again includes unlinkability.

*Clients Result-Security-up-to-Equality Against IS:* Given a sequence of queries (forwarded to IS from QR, possibly by different clients), IS may learn which of the encrypted queries result in the same set of matching documents. No other information about the queries (or the client(s) who generated them) may be learned by IS.

*Client Query-Security-up-to-Equality Against QR :* Given a sequence of queries from a given client, QR may learn nothing beyond which of the encrypted queries are the same.

We are now ready to define our SADS model. Our scheme will satisfy this definition against semi-honest parties, though we will also mention what properties are maintained when some of the parties collaborate, or behave maliciously.

DEFINITION 1. *A Secure Anonymous Database Search (SADS) system consists of protocols for server, client, IS and QR, satisfying all the correctness, security, anonymity, and efficiency requirements defined in Section 3.1 (between client and server) and in this section (for IS and QR).*

# 4. BUILDING BLOCK PROTOCOLS

In this section we present the protocols that we use as basic building blocks for our SADS scheme. First we introduce ***reroutable encryption***, which will be used to achieve client anonymity and authorization. Then we introduce ***private key deterministic encryption***, which will allow to protect the query content from the QR and IS, while enabling efficient search. Finally, we will utilize ***Bloom filters*** as a basis for efficent encrypted search structure.

## 4.1 Re-routable Encryption

Re-routable encryption is a new primitive we will use in our system to protect identities, when routing (encrypted) queries from an authorized client to IS, and also when routing the (encrypted) results back to the client. Informally, re-routable encryption is a protocol to send an encrypted message, or some function of the message, from a sender to receiver through a query router QR, such that two security requirements are satisfied. First is the security of the sender's message with respect to QR, and second is the anonymity of the sender with respect to the receiver. The definitions are given below. We note that this primitive may be of independent interest, e.g., in cases where the query router QR is an intermediary in some protocol for secure multi-party computation, where it is allowed to perform computation only on encrypted data and it must interact with the data owners to perform a computation that requires manipulation of the real data.

DEFINITION 2. *A re-routable encryption scheme consists of algorithms* ($GEN$, $ENC$, $ENC\text{-}QR$, $TRANS$, $DEC\text{-}R$)*:*

- $GEN(1^k, Sender, QR, Receiver)$ *outputs three keys* $(sk, qrk_{\{S,R\}}, rk)$ *for the sender, the QR, and the receiver.*
- $ENC(sk, m) = c$ *encrypts* $m$ *with the sender's key.*
- $TRANS(c, S, st_i) = (R, st_{i+1})$ *identifies the receiver of the message coming from* $S$ *based on the inner state* $st_i$ *of QR, and computes the new state of QR.*
- $ENC\text{-}QR(c, qrk_{\{S,R\}}, st_i) = (\overline{c}, st_{i+1})$ *transforms the ciphertext* $c$ *to a message* $\overline{c}$ *for the receiver* $R$*.*
- $DEC\text{-}R(\overline{c}) = \overline{m}$ *extracts the information that was sent to the receiver from the query router.*

DEFINITION 3 (MESSAGE SECURITY). *Let $S$ be a security definition using an adversary $A$ and applicable to a general encryption scheme. Let $R = (GEN, ENC, ENC\text{-}QR, TRANS, DEC\text{-}R)$ be a re-routable encryption scheme. We say that $R$ provides $S$-message security with respect to $QR$ if $(GEN, ENC, DEC\text{-}R \circ ENC\text{-}QR)$ meets $S$ when $A$ is supplemented with $qrk_{\{S,R\}}$.*

This definition is intentionally non-specific, and can be instantiated using different definitions of security for encryption. For our scheme, we will instantiate this definition both with a standard semantic security notion, and with deterministic encryption security notion.

DEFINITION 4 (SENDER ANONYMITY W.R.T. RECEIVER). *Let $Q_0$ and $Q_1$ be two users with keys $q_0$ and $q_1$ respectively. We say that the re-routable encryption scheme $(GEN, ENC, ENC\text{-}QR, TRANS, DEC\text{-}R)$ with a security parameter $k$ preserves the anonymity of the the sender with respect to the receiver if for any polynomial time adversary $\mathcal{A}$ that given $ENC\text{-}QR(ENC_{q_b}(m))$ for $b \leftarrow_R \{0,1\}$ outputs a guess $b'$, the following holds: $|Pr[b = b'] - \frac{1}{2}| < negl(k)$.*

A re-routable encryption scheme is secure if it meets both of the above definitions.

We will now show one method for constructing a re-routable encryption scheme from an encryption scheme that possesses the following group property:

DEFINITION 5 (ENCRYPTION GROUP PROPERTY). *Let $\Pi = (GEN, ENC, DEC)$ be a deterministic private key encryption scheme. We say that $\Pi$ has a group property if the keys for the encryption scheme form a group and for any message $m$ and any keys $k1$ and $k2$ the following holds: $ENC_{k1}(ENC_{k2}(m)) = ENC_{k1 \cdot k2}(m)$.*

CONSTRUCTION 6 (SIMPLE RE-REROUTABLE ENCRYPTION). *Let $\Pi = (GEN', ENC', DEC')$ be an encryption scheme with the group property from Definition 5. We construct a reroutable encryption scheme $(GEN, ENC, ENC\text{-}QR, TRANS, DEC\text{-}R)$ as follows: In $GEN(1^k)$, Sender and Receiver independently run $GEN'$ $(1^k)$ to create $sk$ and $rk$, respectively. Sender, Receiver, and QR then run a secure multiparty computation with $sk$ as input from Sender, $rk$ as input from Receiver, and $qrk = \frac{sk}{rk}$ as output for QR. In the appendix we give an efficient constant round protocol that realizes the required multiparty functionality.*

*In $ENC(sk, m)$, Sender computes $ENC'(sk, m) = c$. In TRANS, QR chooses a Sender, Receiver pair. In $ENC\text{-}QR(qrk, c)$, QR computes $ENC'(qrk, c) = \bar{c}$. In $DEC\text{-}R(rk, \bar{c})$, Receiver computes $DEC'(rk, \bar{c}) = m$.*

THEOREM 7. *Let $\Pi = (GEN', ENC', DEC')$ be an encryption scheme with a group property, satisfying a security definition $S$. The reroutable encryption $(GEN, ENC, ENC\text{-}QR, TRANS, DEC\text{-}R)$ obtained from $\Pi$ using Construction 6 provides $S$-message security and ensures Sender anonymity w.r.t the receiver. (Proof given in appendix.)*

## 4.2 DET-CCA Deterministic Private Key Encryption Scheme

While the standard definitions of security (e.g., [19]) require an encryption scheme to be probabilistic, a deterministic scheme will allow us considerable efficiency gains, while still providing a level of security which is acceptable in our setting (security-up-to-equality). This tradeoff follows the idea introduced by [22], who define deterministic encryption in the public-key setting, and show how to convert a standard (probabilistic) PKE to a deterministic one. We follow the same approach, adapting it to the secret key setting.

We start by defining chosen-ciphertext (CCA) security for deterministic encryption. The adversary $\mathcal{A} = (A_1, A_2)$, defined as in [22], is a pair of polynomial time algorithms that share neither coins nor state and has high min-entropy $\omega(\log(k))$ (this is the case for any adverary if the underlying plaintext domain is dense).

DEFINITION 8 (DET-CCA). *Let $\Pi^{det} = (GEN, ENC, DEC)$ be a private key encryption scheme and $\mathcal{A} = (A_1, A_2)$ be an adversary against it. We conduct the following two experiments:*

| $\boldsymbol{DET\text{-}EXP}^0_{\pi^{det},\mathcal{A}}(n)$ | $\boldsymbol{DET\text{-}EXP}^1_{\pi^{det},\mathcal{A}}(n)$ |
|---|---|
| $s \leftarrow GEN(1^n)$ | $s \leftarrow GEN(1^n)$ |
| $(\boldsymbol{x}_1, t_1) \leftarrow A_1(1^n)$ | $(\boldsymbol{x}_0, t_0) \leftarrow A_1(1^n); (\boldsymbol{x}_1, t_1) \leftarrow A_1(1^n)$ |
| $\boldsymbol{c} \leftarrow ENC_s(\boldsymbol{x}_1)$ | $\boldsymbol{c} \leftarrow ENC_s(\boldsymbol{x}_0)$ |
| $t' \leftarrow A_2^{ENC_s}(1^n, \boldsymbol{c})$ | $t' \leftarrow A_2^{ENC_s}(1^n, \boldsymbol{c})$ |
| $output \begin{cases} 1 \text{ if } t' = t_1 \\ 0 \text{ else} \end{cases}$ | $output \begin{cases} 1 \text{ if } t' = t_1 \\ 0 \text{ else} \end{cases}$ |

*We define the adversary advantage as $\boldsymbol{Adv}^{DET-CCA}_{\pi^{det},\mathcal{A}} = Pr[\boldsymbol{DET\text{-}EXP}^0_{\Pi^{det},\mathcal{A}}(n)=1] - Pr[\boldsymbol{DET\text{-}EXP}^1_{\Pi^{det},\mathcal{A}}(n)=1]$. We say that $\Pi^{det}$ is DET-CCA secure if for all adversaries $\mathcal{A}$ the advantage $\boldsymbol{Adv}^{DET-CCA}_{\pi^{det},\mathcal{A}}$ is negligible.*

Next we give a construction for converting any semantically secure private key encryption scheme into a deterministic DET-CCA secure private key encryption scheme.

CONSTRUCTION 9 (DETERMINISTIC PRIVATE KEY ENC). *Let $\Pi = (GEN, ENC, DEC)$ be any probabilistic private key encryption scheme and let $H$ be a hash function, which we will model as a random oracle. We define a deterministic private key encryption scheme $\Pi^{det} = (GEN', ENC', DEC')$ as follows:*

- $s = GEN'(1^n) = GEN(1^n)$
- $c = ENC'_s(x) = ENC_s(x; H(s, x))$
- $x = DEC'_s(c) = DEC_s(c)$, $r = H(s, x)$; *return $x$ if $ENC_s(x, r) = c$ and $\perp$ otherwise.*

THEOREM 10. *Let $\Pi = (GEN, ENC, DEC)$ be any probabilistic private key encryption scheme and $\Pi^{det}$ be the corresponding deterministic scheme according to Construction 9. Let $\mathcal{A} = (A_1, A_2)$ be a DET-CCA adversary with min-enthropy $\mu$ against $\Pi^{det}$ that outputs vectors of size $v$ and makes at most $q_h$ queries to the hash oracle and $q_d$ queries to the decryption oracle. Let $ms$ and $mc$ be the max secret key and the max-ciphertext probabilities for $\Pi$. The there exists an IND-CPA adversary $\mathcal{B}$ against $\Pi$ such that*

$$\boldsymbol{Adv}^{DET-CCA}_{\pi^{det},\mathcal{A}} \leq \boldsymbol{Adv}^{IND-CPA}_{\pi,\mathcal{B}} + \frac{2q_h v}{2^\mu} + 2q_h ms + 2q_d mc,$$

*where $\mathcal{B}$ makes at most $v$ queries to its oracle and its running time is within $O(q_h(T_\epsilon + q_d))$ and $T_\epsilon$ is the running time for the encryption algorithm.*

This proof follows the proof of Theorem 5 in [22], replacing the (public key) encryptions with calls to an encryption oracle for the private key encryption scheme.

We instantiate the above deterministic private key encryption scheme following the construction of RSA-DOAEP in [22] but with different primitives that give more security and the group property that we need. We use the Pohlig-Hellman (PH) function [23] and the SAEP+ (short for Simple-OAEP) padding construction introduced in [7]. The Pohlig-Hellman function $PH_k(x) = x^k \ mod \ p$ ($p$ is prime and $k$ is a random key) viewed as an encryption function has the group property from Definition 5 since $(\mathbf{Z}_p, *)$ is Abelian group. The SAEP+ padding scheme is a simpler scheme than OAEP [2] and provides better security guarantees. We now define the private key probabilistic encryption PH-SAEP+ and the private key deterministic encryption PH-DSAEP+, both of which will be used in our scheme. Proofs of (CCA and DET-CCA) security are given in the appendix.

DEFINITION 11 (PH-SAEP+ ENCRYPTION). *We define PH-SAEP+ = (GEN, ENC, DEC) in the following way*

- *$GEN(1^n) = (p, k, k')$ where $p$ is a prime that is publicly known and $k$ is a secret key and $k'$ is its inverse, which is efficiently computable.*
- *$c = ENC_k(x) = PH_k(SAEP+(M, r)) = PH_k(((M \ || \ G(M||r)) \oplus H(r)) \ || \ r)$ where $r$ is chosen at random.*
- *DEC(c):*
  1. *Compute $c' = c^{k'} \mod p$ where $c' = c'' \ || \ r$.*
  2. *Extract $r$ and compute $H(r)$.*
  3. *Compute $x \ || \ g = c'' \oplus H(r)$.*
  4. *Verify that $g = G(x||r)$ and return $x$.*

THEOREM 12. *Assume that no algorithm with running time $t$ can solve the discrete log problem with probability more than $\epsilon$. Then PH-SAEP+ is chosen ciphertext secure scheme in the random oracle model satisfying the following:*

$$t' \leq t/2 - O(q_D + q_G + q_H^2)$$
$$\epsilon' \leq \epsilon^{1/2} + q_D/2^{s_0} + q_D/2^{s_1},$$

*for an adversary with running time $t'$ and advantage $\epsilon'$ that issues $q_D$ decryption queries, $q_G$ queries to $G$, $q_H$ queries to $H$ where $s_1$ is the length of the randomness used and $s_0$ is the length of the output of $G$.*

COROLLARY 13 (PH-DSAEP+). *Let PH-DSAEP+ be the deterministic private key encryption scheme derived by applying Construction 9 to PH-SAEP+. PH-DSAEP+ is DET-CCA secure under the discrete log assumption and in the random oracle model.*

## 4.3 Bloom Filters

The deterministic encryption scheme that we presented provides ciphertexts that are suitable to be used in efficient search protocols according to [22]. Bellare et al. in [22] suggest that the search functionality over encrypted data produced with a deterministic encryption should be realized by attaching "tags" that will be easily searchable and easily computed by both the querying party and the server. We realize the "tagging" idea with a Bloom filter [6]. This facilitates efficient search, guarantees there will be no false negatives, and allows a tunable rate of false positives: $(1 - (1 - \frac{1}{m})^{hn})^h \approx (1 - e^{-\frac{hn}{m}})^h$ where $n$ is the number of entries in the Bloom filter, $h$ is the number of hash functions, and the filter is $2^m$ bits long. For the purposes of our scheme we use a Bloom filter (BF) per document in the database to store the encrypted keywords of that document. We modify the computation of the BF indexes corresponding to a keyword $w$. If we have a Bloom filter of size $2^m$ bits with $h$ hash functions, we use as index values into the BF the first $h$ blocks of $m$ bits each of the encryption of $w$. This preserves the same false positive rate as a standard Bloom filter since the ciphertexts have a pseudorandom property.

# 5. SECURE ANONYMOUS DATABASE SEARCH

In this section we present the protocol for secure anonymous database search that realizes the functionality described in Section 3, and analyze its security.

## 5.1 SADS Protocol Description

We now present the SADS scheme that allows a data owner to make its database available for search by computing BF structures for encrypted search on it and sending them to a index server, which executes queries submitted to it anonymously by authorized queriers via a query router. We use two instantiations of re-routable encryption: for query submission, we instantiate with (DET-CCA secure) PH-DSAEP+, where the QR computes the first BF indices of the encrypted query before passing them on to IS. For returning the results, we instantiate with (IND-CCA secure) PH-SAEP+ directly accordhing to Construction 6.

CONSTRUCTION 14 (SADS). *Let us have a server(S), a client (C), a query router (QR) and an index server (IS). Let $FP$ be the upper bound on the false positive rate that we allow for search. Let $h$ and $m$ be parameters computed based on the sizes of the documents in the database of S such that a Bloom filter of size $2^m$ bits using $h$ hash functions with as many entries as the largest document in the database allows at most false positive rate of $FP$.*

*Let ($GEN_{query}$, $ENC_{query}$, $TRANS_{query}$, $ENC\text{-}QR_{query}$, $DEC\text{-}R_{query}$) be an instantiation of Simple re-routable encryption (Construction 6) with the encryption scheme PH-DSAEP+ and the difference that $ENC\text{-}QR_{query}(c, qrk_{\{S,R\}}, st_i)$ = $BF(PH\text{-}DSAEP+_{qrk_{\{S,R\}}}(c)) = BF(\bar{c}) = \{b_1, \ldots, b_i\}$, where $\{b_1, \ldots, b_i\}$ are the BF indexes obtained from the first $h$ $m$-bit blocks of $\bar{c}$ and $DEC\text{-}R_{query}$ is the identity function.*

*Let ($GEN_{result}$, $ENC_{result}$, $TRANS_{result}$, $ENC\text{-}QR_{result}$, $DEC\text{-}R_{result}$) be an instantiation of Simple re-routable encryption (Definition 6) with the encryption scheme PH-SAEP+.*

**Preprocessing:** *S generates for each of its documents a Bloom filter from the encryptions of its stemmed keywords under PH-DSAEP+ with key $rk_S$ according to the BF modification from Section 4.3. S sends the resulting Bloom filters to IS.*

**Key Generation:** *To authorize C for search S, QR and C run $GEN_{query}$ to obtain keys $(sk_C, qrk_{\{C,S\}}, rk_S)$ for query submission, where $rk_S$ is the key S used in the previous step. Also IS, C and QR run $GEN_{result}$ to get $(sk_{IS}, qrk_{\{IS,C\}}, rk_C)$ that will be used later for the result return.*

**Query Submission:** *To submit an encrypted query for keyword $W$ C computes $c_1 = ENC_{query}(sk_C, W)$ and sends it to*

*QR, which computes* $TRANS_{query}(c_1, C, st'_i) = (IS, st'_{i+1})$ *and* $\{b_1, \ldots, b_i\} = ENC\text{-}QR_{query}(c, rqk_{\{C,S\}}, st'_i)$ *and sends* $\{b_1, \ldots, b_i\}$ *to IS.*
**Search:** *IS obtains* $DEC\text{-}R_{query}(\{b_1, \ldots, b_i\}) = \{b_1, \ldots, b_i\}$ *and uses them for BF search to get the result R.*
**Query Return:** *IS encrypts and sends* $c_2 = ENC_{result}(sk_{IS}, R)$ *to QR. Then QR transforms* $TRANS_{result}(c_2, IS, st''_i) = (C, st''_{i+1})$ *and sends* $\overline{c} = ENC\text{-}QR_{result}(c_2, rqk_{\{IS,C\}}, st''_i)$ *to C, which decrypts* $DEC\text{-}R_{result}(rk_C, \overline{c})$ *to obtain the result R.*

## 5.2 Protocol Security

We now prove the security of the protocol in our semi-honest adversarial model, and then discuss what properties may be compromised if some parties collaborate or act maliciously.

THEOREM 15. *The SADS protocol of Construction 14 satisfies Defintion 1 for semi-honest index server and query router, under the discrete log hardness assumption and in the random oracle model.*

PROOF. The *correctness* of the protocol follows from the fact that BF search does not yield false negatives and we can make the false positive rate arbitrarily small by choosing the appropriate size and number of hash functions for the BF based on its capacity. We consider a false positive rate that can be controlled to be under any acceptable threshold to be a sufficient privacy guarantee, for example, fixing FP rate to a value comparable to the probability of finding a collision in a hash function. Viewing the search functionality as part of the security of a whole system such a FP rate will be well below the error rate of any other part of the system [20]. *Server security* follows from the fact that the only data-dependent information IS sends (and the client receives) consists of the BF indices, namely the intended output. Since the original data owner participate only in the preprocessing and not the actual encrypted search protocol, he does not learn anything about the queries submitted by clients, thus *client security* is achieved. Since all the information IS has about the data is encrypted, *data security against IS* is maintained. The semi-honest QR controls the authorization of clients, providing *server access control*.

Since the query submission and query return are done through re-routable encryption, we may use Theorem 7. For the query submission the sender is the client, the receiver is IS, and the underlying security notion is DET-CCA. Thus, Theorem 7 gives *client anonymity against IS*, as well as DET-CCA message security, which guarantees *client query-security-up-to-equality against QR. Client result-security-up-to-equality against IS* follows from the fact that IS has the power of an adversary in the PH-DSAEP+ scheme, and from the security guarantee of the det-CCA encryption. For the query return the sender is IS, receiver is the client, and the underlying security notion is standard IND-CCA. Thus, Theorem 7 directly implies *data security against QR.*

Finally, in terms of *efficiency*, the scheme requires constant communication for query submission and thus depends only on the size of the return set. Computation complexity grows only with the number of documents, and not with the size of the documents, as it requires a BF search per document, regardless of size. □

As stated in the theorem, our adversarial model assumes QR and IS are semi-honest and do not collude with each other or other parties. We now briefly examine what happens when this assumption is violated. First, note that if IS and QR collude, the client's security properties are maintained, but the client's anonymity (and unlinkability) is violated (for example, they can detect when different users submit the same queries or queries that have the same result). If IS and QR further collude with the server or the client, the security guarantees for the other party are completely compromised If QR colludes with the client, while IS is semi-honest, this may allow an unauthorized client to submit a query, but no other requirements are violated. Finally, if IS collaborates with the data owner, some information about the query may leak (e.g., the first BF indices of the encrypted query).

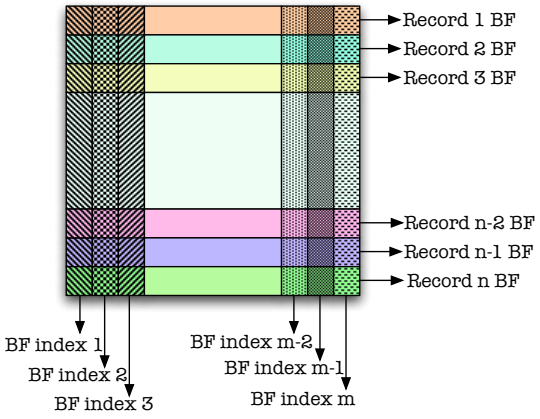



**Figure 2: Multiple Bloom Filters Memory Storage**

# 6. EFFICIENT STORAGE AND EFFICIENT BLOOM FILTER SEARCH

To minimize the number of bits that need to be read to satisfy queries across a large number of Bloom filters, we store them in transposed order. First, they are divided into blocks of filters; within each block, all bits from a single index across the filters are stored sequentially. Thus, each document is represented by a bit within multiple slices, one for each index of its Bloom filter representation (Fig. 2). To run a query, we need only fetch those slices which correspond to the indices of the query term, which is a large savings since normally we would have to read the full contents of every Bloom filter for every document for any query. This technique is referred to as bitslicing and has been studied as a method for storing signature files in database indexes [32].

## 6.1 Slicing optimizations

By storing the Bloom filters in blocked slices, we gain the ability to avoid reading a large portion of the bits in the Bloom filter set when we run queries. We need only check those slices which correspond to an index which is present in the query term(s). Since this is very sparse, this is a large improvement over non-transposed storage; it would require us to read the entirety of every Bloom filter in order to run a query.

To run a query, we construct a result vector, which is a bit vector equal in size to the number of Bloom filters in the

set. This is then "and"ed to each slice corresponding to a query index. Over time, several block-sized portions of the result vector will become zeroed out. Once this happens, as a further optimization we cease to read those portions of later indices. Our block size is chosen as the disk page size, and our end goal is thus to read the minimum number of pages necessary to answer a query. If multiple queries are being run, we keep a cache of recently viewed bitslices with a LRU replacement policy.

Because we are storing the Bloom filters in transposed order, and each filter is represented by a single bit across various slices, deletion of filters would be expensive. Thus, we implement this simply by zeroing out the indices of a filter so that it will not match future queries. As a future addition, we may support a system of periodically cleaning the slicebase by identifying "deleted" filters and compacting the remaining ones.

## 6.2 Boolean queries

Supporting AND queries is trivial; the Bloom filter indices of the query terms can simply be unioned before running the query indices across the set as if they were a single term. Supporting OR queries is more difficult, since we must know the results of each individual query before we can union them. There is no operation we can do on the query indices to achieve this with a single query.

However, there are still optimizations available to us. In order to run an OR query with $x$ terms, we must generate $x$ result vectors and run them over the entire set. Rather than run them individually, we run them in parallel. This not only improves the cache behavior of our bitslice fetching, but also allows us to avoid reading blocks from later slices once we see that the corresponding blocks of all $x$ result vectors have been zeroed out by earlier indices. In order to increase the likelihood of this happening, we query the indices in order of the number of distinct query terms they appear in.

This technique for running OR queries is sufficient to address any boolean query which can be represented with a monotone disjunctive normal form. Disjunctive normal form requires that the query be phrased as a disjunction (an OR clause) of one or more conjunctions (an AND clause) of literals. We can represent conjunction queries by unioning the indices that represent the literals within, and then pass the disjunction of those conjunctions to our OR query method.

# 7. PERFORMANCE

We implemented and tested our secure anonymous search with various parameters. Unfortunately, we cannot provide a direct performance comparison to existing systems for a number of reasons. Firstly, existing protocols for encrypted search address scenarios different from ours with different threat models. Secondly, few papers have implementations of the schemes they propose. Those that do, have not made their implementations publically available, so we cannot run them on similar hardware. Thus for the sake of comparison, we present analytical comparisons in Table 1.

Note that some schemes guarantee both client and server privacy such as [8, 26, 27], and amongst these some can be adapted to fit our model where the data is owned by the server rather than the querier. We denote this with yes* in the table. These schemes have both high computational and communication complexities. Schemes like [14] and [9] achieve better complexities, however, they assume that the client is the real owner of the data and use this fact to employ techniques not applicable to our scenario. For example the client, knowing the address location of the data, can reorder it, compute inverted indexes for search, retrieve more documents than the ones matching the search in order to guarantee the privacy of the client with respect to the server.

| Protocol | PEKS [8] | Logs [27] | Enc.S. [26] | SSE [14] | PKE PIR [9] | SADS |
|---|---|---|---|---|---|---|
| 1 doc | O(w) | O(w) | O(w) | O(1) | O(w $\log^3$w) | O($\log_{0.5}$ FP) |
| n docs | O(nw) | O(nw) | O(nw) | O(n) | O(Ln) | O($\rho$n) |
| comm. compl. | O(nw) | O(nw) | O(nw) | O(L) | O(poly log(n)) | O(L) |
| client priv. | yes | yes | yes | yes | yes | yes |
| server priv. | yes* | yes | yes* | no | no | yes |

**Table 1: Encrypted Search Schemes Comparison: computational complexity for search on one or n documents each containg w keywords, communication complexity for a single search, the client and server privacy; L is the number of returned matching results and $\rho$ = number of BF hash functions/memory block size $<<$ 1, $0 <$ FP $< 1$ is the expected false positive rate for the Bloom filters.**

Although we do not have implementations of these systems for comparison, we implemented our own system in C++ to demonstrate practicality of use. We ran experiments on a Ubuntu 8.04 Linux PC with a Pentium 4, 3.4 GHz cpu and 2GB of RAM. A variety of corpus sizes from 1K to 50K were extracted from the Enron Email Dataset, available at `http://www.cs.cmu.edu/~enron/`. Each document was stemmed using the techniques provided by the Clair library [24], and the stems were inserted into the Bloom filter index per document. Bloom filter sizes were computed to give a false positive rate of 0.1% based on the number of stems we wished to be able to index.

Table 2 shows the time taken to create indexes for each of our 7 corpuses. The creation time scales roughly linearly with the corpus size as one would expect. Although expensive, these times are a one-time cost for index creation, and can further be parallelized if need be. Disk space requirement was very low, reaching only 128 Kb for the largest corpus we indexed. Thus, both time and storage costs are reasonable for use in real-world applications.

Before running queries, we extracted from the stem set a subset of query terms, and grouped them by document frequency within the database. In each experiment, we ran a total of 100 queries and took the average time to completion for each query. If we had less than 100 terms to query on, we cycled through the existing ones, spacing out identical queries to minimize artificial cache gains. Figure 3 shows

| Size | 1K | 5K | 10K | 20K | 30K | 40K | 50K |
|---|---|---|---|---|---|---|---|
| Time | 10m | 44m | 80m | 158m | 229m | 310m | 397m |

**Table 2: Creation times in minutes for indexes on various corpus sizes**

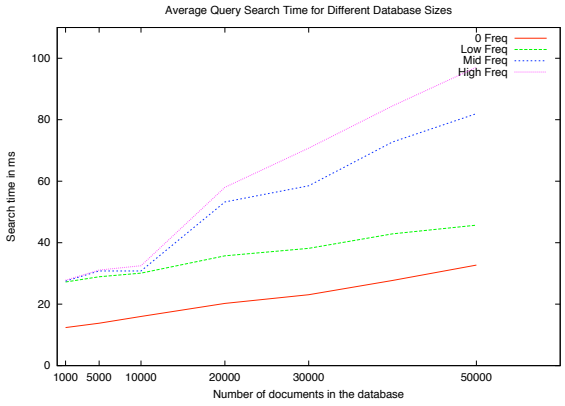



**Figure 3: Search Times**

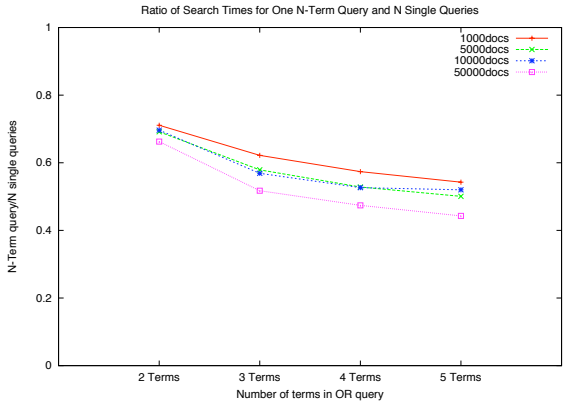



**Figure 5: Ratio of OR Query time vs Individual Queries**

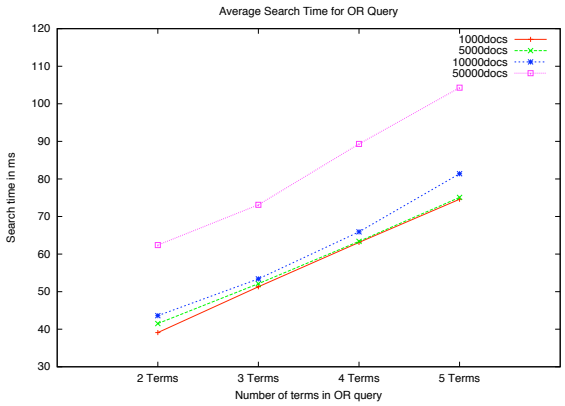



**Figure 4: Time vs OR Query Terms**

the average time per query plotted against the size of the corpus the index was computed from. We further show these curves in relation to four different types of queries based on frequency of the query terms being searched on. The four frequency groups used were: **0-Freq** - terms which do not appear anywhere in the corpus; **Low Freq** - terms which appear in 1 or 2 documents; **Med Freq** - terms which appear in 45-55% of the corpus; **High Freq** - terms which appear in all but 1 or 2 of the documents.

As expected, query time grows roughly linearly with an increase on corpus size, as a larger number of bloom filters must be queried. There is a noticeable hump in the curve for the higher two frequency groups when the corpus size reaches 20K and above. This is likely because a sufficient number of bit slices is being fetched to fulfill these queries that no longer remains entirely in cache.

There are minor improvements when running queries on infrequent terms (in other words, running queries with fewer results). This culminates in a marked improvement for queries which return no results. This is due to the fact that the bit-slice query operation will not need to fetch later bitslices if portions of the result vector already reveal from earlier indices that certain documents are not matches. Once a full block of documents is ruled out, we do not fetch slices representing later indices from those documents. This is most pronounced for queries which have no results.

In this and all of our experiments, larger corpus sizes cause queries to take a little longer, since more Bloom filters and hence more slices must be checked. However, the relatively small time differences compared to the size differences indicates that the time spent running Bloom queries is extremely small compared to other operations such as the exponentiations required to run our encryptions, which do not vary with corpus size.

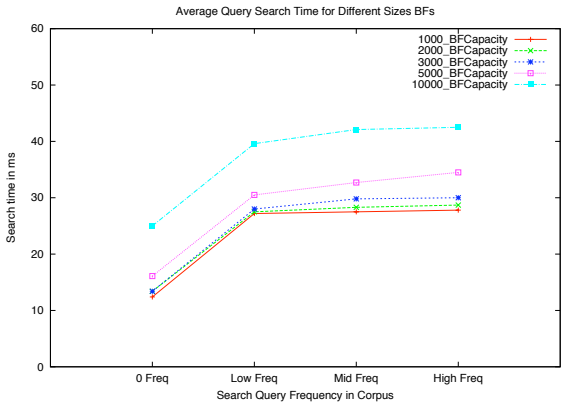
Figure 4 shows the average time per query plotted against OR queries with varying numbers of terms in them. As one would expect, the larger the number of terms being queried, the longer it takes to complete a query. Since each term maintains a distinct result vector, a large number of terms reduces the probability that documents can be ruled out early and prevent us from having to fetch later slices.

Figure 5 shows the average time per query plotted against OR queries as a ratio against the amount of time it would take to run these queries individually and union the results afterwards. As we can see the savings are significant, and grow more so as the number of terms increases. When running these terms in parallel as an OR query, a slice fetched remains in memory and can be checked against each query quickly. When running them separately, they must be fetched multiple times. As we can also see, this effect grows less pronounced with larger corpus sizes, since with smaller corpus sizes there is an increased likelihood that slices will remain cached from previous runs even while running the queries individually.

Figure 6 shows the average time per query while varying the Bloom filter size and keeping the corpus size fixed at 1K. The Bloom filter sizes were calculated to keep the same 0.1% false positive rate we were aiming for before while supporting a varying number of terms to insert into the index per document. As one would expect, with larger Bloom filter sizes, we see an increase in the time per query; however this is still small in relation to the increase in size. Again, this is be-

| | local server | trans-US | Europe |
|---|---|---|---|
| Ping time (ms) | 0.227 | 90.615 | 110.978 |

**Table 3: Ping Latency**




| Capacity | 1k | 2k | 3k | 5k | 10k |
|---|---|---|---|---|---|
| Size(bits) | 16384 | 32768 | 65536 | 131072 | 262144 |

**Figure 6: Time vs Term Frequency (varying BF size)**

cause query time is dwarfed by time spent on cryptographic operations. In general, query times remained below 100ms per query, and are acceptably small in relation to what one would expect for network delays. Table 3 shows network delays for different typical distances around the world as a point of comparison.

Furthermore, on long running tests, analysis via gprof revealed that 64.9% of time was spent running cryptographic operations. For queries, these are run by the client and the query router, not the index server which would be the bottleneck in a system with many queries coming from various sources. This technique should thus introduce an acceptable overhead that makes this a practical system for real world use.

# 8. CONCLUSION

We proposed a solution for the problem of secure anonymous database search, which addresses the issue of allowing untrusting parties to search each other's private data when there are legitimate reasons for this. A major goal for us is to achieve practical efficiency while still achieving the maximal security and privacy guarantees that the efficiency requirement permits. For this purpose we utilized a security architecture with distributed limited trust among two intermediary parties, which we consider viable in practical situations where we have authorities regulating the controlled data sharing without learning any private information of the participants. We defined a new primitive, re-routable encryption, that we use to implement interactions between the participants in the encrypted search protocol. To achieve the protocol efficiency we use Bloom filters as search structures on the ciphertexts produced from the database keywords with a new private key deterministic encryption, PH-DSAEP+, necessary to overcome the linear bound on search complexity. We implement and study the practical performance of our protocol which demonstrates search times comparable to average network delay, which we consider an appropriate measure for usable efficiency.

# 9. REFERENCES


[1] Adam J. Aviv, Michael E. Locasto, Shaya Potter, and Angelos D. Keromytis. Ssares: Secure searchable automated remote email storage. *Computer Security Applications Conference, Annual*, 0:129–139, 2007.

[2] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption – how to encrypt with rsa. In *Proceedings of EUROCRYPT'94*, 1995.

[3] Steven M. Bellovin and William Cheswick. Privacy-enhanced searches using encrypted bloom filters. Technical Report CUCS-034-07, Department of Computer Science, Columbia University, September 2007.

[4] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Proceedings of EUROCRYPT'98*, 1998.

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

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

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

[8] 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.

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

[10] 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.

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

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

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

[14] 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.

[15] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. Rsa-oaep is secure under the rsa assumption. *J. Cryptol.*, 17(2):81–104, 2004.

[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] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM.

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

[20] Laura J. Heath. An analysis of the systemic security weaknesses of the U.S. Navy Fleet Broadcasting System, 1967–1974, as exploited by CWO John Walker. Master's thesis, U.S. Army Command and General Staff College, 2005.

[21] Markus Jakobsson, Ari Juels, and Paul Syverson. Universal re-encryption for mixnets. In *In Proceedings of the 2004 RSA Conference, CryptographerÕs track*, pages 163–178. Springer-Verlag, 2004.

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

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

[24] Dragomir R. Radev, Mark Hodges, Anthony Fader, Mark Joseph, Joshua Gerrish, Mark Schaller, Jonathan dePeri, and Bryan Gibson. Clairlib documentation v1.03. technical report cse-tr-536-07. *University of Michigan. Department of Electrical Engineering and Computer Science*, 2007.

[25] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 350–364, Washington, DC, USA, 2007. IEEE Computer Society.

[26] 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.

[27] B. Waters, D. Balfanz, G. Durfee, and D. Smetters. Building an encrypted and searchable audit log. In *NDSS 2004.*, 2004.

[28] Peter Williams and Radu Sion. Usable pir. In *NDSS 2008.*, 2004.

[29] Peter Williams, Radu Sion, and Bogdan Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 139–148, New York, NY, USA, 2008. ACM.

[30] Andrew Chi-Chih Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[31] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[32] Justin Zobel and Alistair Moffat. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23:453–490, 1998.

# APPENDIX

## A. RE-ROUTABLE ENCRYPTION

Proof of Theorem 7. First we show that the obtained reroutable encryption provides $S$-message security. Assume that (GEN, ENC, DEC-R $\circ$ ENC-QR) does not meet the security definition $S$. Therefore there exists an adversary $\mathcal{A}$ that can obtain information $t$ about a message $m$ given $\mathrm{ENC}(m)$. Since $\mathrm{ENC}(m) = \mathrm{ENC}'(m)$ it follows that $\mathcal{A}$ is an adversary against $\Pi$ that manages to learn information $t$ about an encrypted message from its ciphertext, which is a contradiction with the security property of $\Pi$.

Second we prove that the reroutable encryption ensures Sender anonymity w.r.t the receiver. Let $q_0$ and $q_1$ be the keys of two senders and $qr_0 = \frac{rk}{q_0}$ and $qr_1 = \frac{rk}{q_1}$ be the corresponding transformation keys at the third party. Let $m$ be any message. Now using the group property of $\Pi$ we have

$$\mathrm{ENC\text{-}QR}(\mathrm{ENC}_{q_0}(m)) = \mathrm{ENC}_{qr_0}(\mathrm{ENC}_{q_0}(m)) =$$
$$= \mathrm{ENC}'_{qr_0}(\mathrm{ENC}'_{q_0}(m)) = \mathrm{ENC}'_{qr_0 \cdot q_0}(m) = \mathrm{ENC}'_{rk}(m)$$

$$\mathrm{ENC\text{-}QR}(\mathrm{ENC}_{q_1}(m)) = \mathrm{ENC}_{tp_1}(\mathrm{ENC}_{q_1}(m)) =$$
$$= \mathrm{ENC}'_{qr_1}(\mathrm{ENC}'_{q_1}(m)) = \mathrm{ENC}'_{qr_1 \cdot q_1}(m) = \mathrm{ENC}'_{rk}(m)$$

Therefore the index server will always get the same ciphertext and cannot guess the user identity with probability non-negligibly different from 1/2. □

## B. DET-CCA PRIVATE KEY ENCRYPTION

Proof of Theorem 12 and Corollary 13 . In [7] Boneh shows how to construct a CCA secure public key encryption from a trapdoor permutation and the SAEP+ scheme. We modify this technique to obtain a CCA secure private key encryption scheme.

The idea of public key scheme is captured in the functionality of a trapdoor permutation. The definition of a trapdoor permutation is a function $f$ which is easy to evaluate, hard to invert on its own, but easy to invert with the knowledge of some trapdoor information. Now we consider the Pohlig-Hellman function from Section 4. The value $PH_k(x)$ is hard to invert without knowledge of $k$ because of the hardness assumption of the discrete logarithm problem. On the other hand, given $k$ and $PH_k(x)$ it is easy to compute the inverse $k^{-1}$ and find $m$. The only thing left to be able to view $PH_k(x)$ as a trapdoor permutation is to provide a way to compute easily without knowing $k$. We can achieve this by queries to an oracle $O^{PH_k}$ that implements the functionality of $PH_k(x)$.

Boneh ([7]) defines the set partial one-wayness problem to find a set of values that contains the inverse of a given value produced by a trapdoor function $f$ and connects the security of the $f$-SAEP+ to the hardness of solving the onewayness problem. We translate this result to the case of the Pohlig-Hellman function.

Definition 16 (PH Set Partial One-wayness Problem) *Let $PH_k(x)$ be the Pohlig-Hellman function that is modeled as a random oracle $O^{PH_k}$ and $k$ is secret. We say that an algorithm $\mathcal{A}$ solves the set partial one-wayness problem $(PH_k(x), r)$ if given $c = PH_k(x)$ it produces a set $S = \{x_1, \ldots, x_r\}$ such that $c = PH_k(x_i)$ for some $1 \leq i \leq r$.*

Lemma 17. *Let $\mathcal{A}$ be a $(t, q_D, q_H, q_G)$ chosen ciphertext attack algorithm in the random oracle model where $\mathcal{A}$ runs in time $t$, makes $q_H$ queries to the oracle $H$, $q_G$ queries to the oracle $G$ and $q_D$ decryption queries of PH-SAEP+ and has an advantage $\epsilon$. Then there exists an uniform algorithm $\mathcal{B}$ that solves the set partial one-wayness problem $(PH_k(x), q_H)$ with the following time and advantage:*

$$time(\mathcal{B}) \leq time(\mathcal{A}) + O(q_H + q_G + q_D)$$
$$adv(\mathcal{B}) \leq adv(\mathcal{A})(1 - q_D/2^{s_0} - q_D/2^{s_1})$$

The proof of the Lemma 17 is analogous to the proof of Theorem 5 in [7] where the evaluation of the trapdoor function $f$ are substituted with calls to the random oracle $O^{PH_k}$ implementing the Pohlig-Hellman function.

We now have all the necessary tools for the theorem proof. Let us assume that there is a $(t, q_D, q_H, q_G)$ chosen ciphertext adversary $\mathcal{A}$ against PH-SAEP+ with advantage $\epsilon$. By Lemma 17 we know that there is a $t'$-time adversary $\mathcal{B}$ that solves the PH set partial one-wayness problem $(PH_k, q_H)$ with advantage $\epsilon'$ for some $t'$ and $\epsilon'$. Fujisaki et al. in [15] demonstrate an algorithm that runs $\mathcal{B}$ twice on $C^*$ and $\alpha C^*$ for some $\alpha$ and uses the resulting sets $S$ and $S_\alpha$ to compute the $k$-th root of $C^*$ in time $O(q_H^2)$ and hence breaks the $PH$ with probability $\epsilon'^2$. The theorem now follows. And Corollary 13 also follows. □

# C. KEY GENERATION FOR OUR SADS PROTOCOL

The key generation algorithm is not our main focus since general multiparty computation techniques ([18,30,31]) can be applied to distribute the appropriate keys. However, we give here an efficient algorithm that allows the sender (S), the receiver (R) and the query router (QR) to obtain their keys. The sender and the receiver choose their keys $k_S$ and $k_R$ respectively. The sender chooses a random number $r_S$ and the receiver chooses a random number $r_R$; the following messages are exchanged between the three parties using a public key encryption scheme $(\text{GEN}, \text{ENC}, \text{DEC})$ in which $pk_{QR}$ and $pk_S$ are public encryption keys for the third party and the sender.

$$\begin{aligned} S \rightarrow QR &: k_S \cdot r_S \\ R \rightarrow QR &: k_R \cdot r_R \\ R \rightarrow S &: r_R \\ S \rightarrow QR &: r_S \cdot r_R^{-1} \end{aligned}$$

At the end of the above message exchange the query router can compute: $k_{QR} = (k_R \cdot r_R) \cdot (k_S \cdot r_S)^{-1} \cdot r_S \cdot r_R^{-1} = (k_R) \cdot (k_S)^{-1}$.

In the above protocol a misbehaving party can cause at most invalid third party key but it cannot learn any secret. Our adversarial model assumes no colluding parties during key generation. If the query router colludes with the sender or the receiver, they can compute from their keys the key of the non-colluding party. Collusion between sender and receiver is not possible since these parties want to protect their data from each other.